heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

Hes·so
Haute Ecole Spécialisée
de Suisse occidentale

Projet de Diplôme Bachelor

ISCIA : identification et qualification des adresses IP sources de spam frauduleux

Auteur : Nicolas Muller

Date : 16 décembre 2010
Module : PDB
Filière : TIC
Proposé par : Patrick Mast, projet Ra&D ISCIA
Professeur responsable : Christian Buchs

heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

Département **TIC**
Filière **Informatique**
Candidat **Muller Nicolas**

TRAVAIL DE DIPLOME 2010

ISCIA : identification et qualification des adresses IP sources de spam frauduleux

---

Proposé par : Patrick Mast, projet Ra&D ISCIA

Résumé du problème :

Sur Internet, de plus en plus de messages sont envoyés par des malfrats dans le but de s'enrichir illégitimement : escroqueries (p. ex. "Nigerian Connection"), arnaques, contrefaçons, phishing et autres délits par emails. Des études préalables ont permis d'établir certaines caractéristiques de ces messages et ont confirmé la présence de leurs auteurs dans plusieurs pays européens. Dans le but d'aider le renseignement criminel, le projet ISCIA (Internet Surveillance for Criminal Intelligence Analysis) propose (entre autres) un système d'acquisition et d'analyse de ces spams. Ce travail a comme objectif de concevoir et de réaliser une partie de ce système qui permettra d'identifier de manière plus efficace puis de "qualifier" les adresses IP sources à l'origine de ces messages malveillants. Après avoir analysé les besoins et effectué l'état de l'art des techniques permettant d'identifier (p. ex. moteurs de spam) et de "qualifier" les adresses IP actives sur Internet (listes noires de zombies, de serveurs piratés, etc.), il s'agira de concevoir et développer en Java un module présentant une simple interface mais effectuant en amont toutes les recherches nécessaires (p. ex. analyses, maintenance de bases de données locales, etc.).

Hes·so
Haute Ecole Spécialisée
de Suisse occidentale

Cahier des charges :

Le module à réaliser, que nous nommerons "EmailHeaderQualif", fournit un mécanisme modulaire d'analyse d'en-tête de courriel et de qualification via une API. Il permet d'obtenir différentes informations sur les en-têtes. Ce système modulaire doit être extensible. C'est-à-dire que des nouveaux modules d'analyse doivent pouvoir être ajoutés à EmailHeaderQualif par la suite et de façon simple. On entend par simple une intervention quasi nulle dans le code existant. Idéalement il devrait suffire de créer et de compiler la classe correspondant à un module et de déposer le fichier de la classe compilée au bon endroit pour que le module en question soit pris en compte. Cet idéal est évidemment difficilement atteignable mais il faut essayer de s'en rapprocher au maximum.

Il en découle plusieurs tâches listées ci-après :
- Créer le système de traitement modulaire
- Créer les modules d'extraction des informations des en-têtes en modifiant le travail de R. Cornu pour extraire les adresses IP
- Trouver des sources d'information consultables qualifiant les adresses IP (listes noires)
- Créer les modules de recherche d'informations grâce à ce qui a pu être extrait précédemment (via "whois")

Il s'agit d'intégrer le travail de R. Cornu dans le projet. Ce travail comprend un moyen d'extraction d'adresses IP, un système de vérification de la validité d'une adresse IP, la géo-localisation d'une adresse IP et l'obtention de l'agent utilisateur (client utilisé pour écrire et envoyer un courriel) à partir d'un en-tête.

Le couplage entre le projet ISCIA et EmailHeaderQualif doit être minimal. C'est-à-dire que le seul élément de couplage est la signature de l'API. Aucun autre élément ne doit venir en augmenter le couplage (notamment ne pas retourner de types ou d'objets qui auront été créés spécifiquement pour le projet).

Attention toutefois : il n'est pas prévu d'évaluer la pertinence de la qualification des adresses IP. C'est à dire qu'il n'est pas prévu de pouvoir dire, par exemple, "l'adresse IP xxx.xxx.xxx.xxx est un botnet avec une probabilité de y%". En effet il serait trop complexe de mettre en place un tel système d'évaluation cohérant durant le temps imparti à ce travail. En définitive, ce qui sera retourné, ce sont les informations suffisamment fiables qui peuvent être obtenues de l'en-tête.

Traitement des en-têtes
==========

Les en-têtes fournis peuvent contenir plusieurs informations. Le travail de R. Cornu faisant déjà de l'extraction d'adresses IP, la détection de l'agent utilisateur et la géo-localisation,
il devra être intégré sous forme de modules dans le projet EmailHeaderQualif et adapté aux besoins de celui-ci. L'extraction d'adresses IP devra être adaptée pour extraire toutes les adresses IP trouvables et pertinentes dans une investigation (IPs publique et privée de la machine émettrice et IP publique du premier MTA).

Les informations sur les adresses IP ainsi extraites peuvent être obtenues par deux moyens. Le premier étant des serveurs fournissant des listes noires. Le deuxième étant les bases de données telles que RIPE qui stockent des informations sur les différentes adresses IP et intervalles ainsi que sur les propriétaires de ces adresses. Ces bases peuvent être consultées à l'aide de la commande "whois".

Il en découle deux modules réalisables : un module de qualification par liste noires (IPBL) et un module d'informations IP par "whois".

Il est important pour l'utilisateur qu'il puisse estimer la pertinence de ces informations. C'est pourquoi toutes les ressources utilisées pour chacun de ces modules (listes des serveurs consultés et de l'organisme qui les gère) seront listées et détaillées. De plus, les limites de ces informations lors de l'interprétation des données seront expliquées.

Plus globalement, une API de traitement de courriel permettant de qualifier les en-têtes sera réalisée. Cette API fournit une méthode prenant en paramètre un en-tête structuré et la liste des en-têtes "Received". Ces derniers (en particulier ceux introduits par les deux premiers MTA) contiennent potentiellement l'adresse IP privée, l'adresse IP publique de la machine ayant envoyé le courriel et l'adresse IP du premier MTA (Mail Transfert Agent). En retour on obtient l'agent utilisateur, les adresses IP privée et/ou publique de la machine ayant envoyé le courriel et du premier MTA. L'adresse IP publique de la machine ayant envoyé le courriel est qualifiée (voir ci-dessous pour les différentes qualifications) et celle du MTA aussi.

Il est clair qu'il n'est pas toujours possible de trouver ces informations, auquel cas elles ne sont tout simplement pas retournées. Il ne faut pas générer d'erreurs lors d'une telle impossibilité.

La méthode prend en paramètre l'ensemble des "Received" pour permettre une extensibilité des modules sans devoir pour autant modifier la signature de la méthode.
La signature de la méthode est la suivante :
headerQualif (header Map<Srtring><String>, receiveds List<Map<String><String>>)

La liste des "Received" se donne dans une "List". Le premier élément retourné par cette "List"

est le dernier "Received" inséré dans l'en-tête.
Liste des valeurs retournées :
- UserAgent : agent utilisateur utilisé pour envoyer le courriel
- PrivateIP : adresse IP privée de la machine ayant envoyé le courriel
- PublicIP : adresse IP publique de la machine ayant envoyé le courriel
 * [PublicIPQualification1] Première qualifications de l'IP
 * [PublicIPQualification2] Deuxième qualifications de l'IP
 * [PublicIPQualificationX] Xème qualifications de l'IP (où X est un nombre plus grand que 1), le nombre de qualifications n'est donc pas limité.
 * [PublicIPNetName] Nom de l'organisation à laquelle appartient l'IP publique
 * [PublicIPRange] Intervalle dans lequel se trouve l'IP publique
 * [PublicIPcountryCode] Code du pays où se trouve l'adresse IP publique
 * [PublicIPcountryName] Nom du pays où se trouve l'adresse IP publique
 * [PublicIPzipcode] Code postal où se trouve l'adresse IP publique
 * [PublicIPcityName] Nom de la ville où se trouve l'adresse IP publique
 * [PublicIPlongitude] Longitude où se trouve l'adresse IP publique
 * [PublicIPlatitude] Latitude où se trouve l'adresse IP publique
 * [PublicIPmetroCode] Le code du métro (États-Unis uniquement)
 * [PublicIPaeraCode] L'indicatif téléphonique où se trouve l'adresse IP publique (États-Unis uniquement)
 * [PublicIPdmaCode] Le code du marché désigné (DMA) de l'adresse IP publique (États-Unis uniquement)
 * [PublicIPtimeZone] Le fuseau horaire où se trouve l'adresse IP publique
- FirstMTAIP : adresse IP du premier MTA ayant relayé le courriel
 * [FirstMTAIPQualification1] La qualifiation peut aussi être appliquée au premier MTA (par exemple relais ouvert ou proxy anonymiseur)
 * [FirstMTAIPQualification2] Deuxième qualification du MTA
 * [FirstMTAIPQualificationX] Xème qualifiations du MTA (où X est un nombre plus grand que 1), le nombre de qualifications n'est donc pas limité.

Qualifications
==========

On entend par qualification le fait de pouvoir dire quelle est l'origine du mail. Par exemple : l'envoi par une machine zombie. Une qualification peut aussi être vue comme un marqueur. Par exemple un courriel peut avoir été envoyé via un zombie connu comme un ordinateur envoyant des courriels de phishing. On a donc ainsi deux qualifications pour une même origine, le zombie et le phishing. Le nom du webmail (ou user agent) utilisé pour envoyer un courriel peut aussi être vu comme un de ces marqueurs. Les MTA peuvent aussi être qualifiés, notamment

comme relais ouverts ou comme proxy anonymiseur.

Les qualifications possibles sont les suivantes (liste sujette à changements durant le projet) :
- zombie : Ordinateur quelconque infecté
- phishing : Ordinateur connu comme ayant envoyé des courriels de phishing
- knownSpamSender : Ordinateur connu comme envoyant des spams
- anonymiserProxy : Proxy anonymiseur
- openRelay : Ordinateur connu comme étant un relai ouvert

Méthodologie de développement
==========

Le système étant complétement séparé de ISCIA, il faut développer un programme de test pour pouvoir tester le projet.

Le projet comprend trois phases, chacune d'entre elles comprend :
- une partie initiale de spécification comprenant l'établissement des objectifs de la phase et des tests unitaires et fonctionnels à effectuer
- une conception durant laquelle le travail à réaliser est modélisé
- le codage de la phase
- les tests unitaires du code réalisé
- les corrections à faire découlant de ces tests

Ces trois phases sont :
1. Développement de l'analyseur modulaire en y intégrant le travail de R. Cornu
2. Développement du module de recherche d'informations sur les adresses IP (whois)
3. Si le temps le permet, module de qualification des IP par listes noires (IPBL)

Chaque phase comporte un délai précis.

Deux validations seront faites par le mandant et le professeur responsable. La première pour l'analyseur modulaire et la seconde pour les deux modules whois et IPBL.

L'intégration de EmailHeaderQualif dans ISCIA se fera après le travail de diplôme.

Technologies
=========

ISCIA utilise principalement les technologies Java pour le codage, jUnit pour les tests, log4j

pour la journalisation et Maven pour la gestion de projet. Pour permettre une intégration optimale dans le projet, le code fourni doit être compatible avec ces technologies.

Délivrables
==========

- Documentation complète du projet au format PDF. Les sources ayant permis de créer ce PDF (en LaTeX) sont également à fournir
- Code source, en anglais
- Documentation de développement
- Manuel d'installation et d'administration des logiciels réalisés
- Documentation du code (javadoc)
- Programmes de tests et leurs sources, avec un mode d'emploi (en annexe dans la documentation)
- Liste des tests effectués (en annexe)

Lors des évaluations intermédiaire et finale du projet de bachelor, une importance toute particulière sera donnée au respect des directives suivantes :
1. Avant chaque réalisation importante, la ou le diplômant(e) devra s'assurer que les spécifications et la conception de la phase correspondante ont été validées par le professeur responsable (et le mandant le cas échéant).
2. La ou le diplômant(e) est tenu d'effectuer un suivi régulier de son projet et d'en informer le professeur responsable (suivi de l'avancement, résumé du journal de travail, problèmes importants rencontrés, etc.) au moins une fois par semaine.
3. Le rapport doit être rédigé au fur et à mesure de l'avancement du projet. Conformément aux directives du département TIC de la HEIG-VD, une version intermédiaire du rapport final sera soumise et évaluée par le prof. responsable à la fin du deuxième semestre.
4. Le rapport doit contenir des informations détaillées concernant la planification du projet de bachelor, à savoir : les différentes phases du projet et les résultats attendus, les coûts estimés en jours homme de chaque phase et de chaque tâche. Le rapport doit aussi contenir les mêmes informations concernant le travail effectué durant le projet de bachelor : phases et coûts réels. Une comparaison entre les informations planifiées et effectuées est aussi incluse dans le rapport.
5. La présentation lors de la défense du projet de bachelor et le contenu du rapport de projet de bachelor doivent rigoureusement respecter les exigences décrites dans les directives émises par le département TIC de la HEIG-VD.
6. Toute contribution de tiers (code, images, texte, idée importante, etc.) provenant d'Internet ou de n'importe quelle autre source doit être explicitement indiquée et référencée. Tout manquement à cette règle est considéré comme grave et peut entraîner un rejet du projet de

bachelor.

7. Le rapport et la présentation doivent clairement décrire le raisonnement et les démarches entreprises par la ou le diplômant(e) au cours de son projet de bachelor, depuis le cahier des charges jusqu'aux résultats finaux.

8. Le rapport doit contenir toutes les indications requises pour un éventuel développement futur du travail réalisé par de tierces personnes.

9. Dans la mesure du possible, le travail de développement doit suivre les bonnes pratiques reconnues internationalement en matière de qualité et de sécurité IT. Les bonnes pratiques utilisées doivent être référencées et leur utilisation/application clairement décrites.

10. Afin que les experts et le professeur responsable puissent, s'ils le désirent, auditer et tester le travail réalisé, tous les fichiers nécessaires (exécutables, sources, configuration, scripts, etc.) à la régénération (déploiement, recompilation, etc.) du travail réalisé doivent être joints au rapport sur un CD ou DVD (dans une fourre annexée dans le rapport) avec la procédure d'installation nécessaire et un mode d'emploi suffisamment précis. Si un accord de confidentialité et/ou de droit de propriété s'applique sur le projet de bachelor, les documents appropriés et signatures nécessaires doivent être obtenues par la ou le diplômant(e) avant l'envoi du rapport final et du CD/DVD.

11. Le rapport en format Word (ou OpenOffice ou LaTeX) ainsi que la présentation en format PowerPoint (ou OpenOffice) doivent être mis à disposition sous forme de CD/DVD lors de la défense du projet de bachelor. Ce CD/DVD doit également contenir les fichiers décrits au point précédant. Si un accord de confidentialité et/ou de droit de propriété s'applique sur le projet de bachelor, les documents appropriés et signatures nécessaires doivent être obtenues par la ou le diplômant(e) avant la remise de ce CD/DVD.
12. Si aucun accord de confidentialité ne s'applique sur le projet de bachelor et en accord avec le mandant, un aperçu du travail réalisé et de ses caractéristiques pourra être publié sur Internet. Le cas échéant, cette publication devra être présentée lors de la défense du projet de bachelor.

Le candidat :

Muller Nicolas

Le professeur responsable :

Buchs Christian

Le chef du département TIC :

Eduardo Sanchez

Yverdon-les-Bains, le 19 septembre 2010

Résumé

ISCIA est un projet pour l'analyse d'e-mails pour la recherche de fraudes par e-mail. Ce document présente un projet de diplôme Bachelor qui permet à ISCIA d'obtenir, à partir des en-têtes contenues dans un e-mail, les adresses IP pouvant fournir des informations pertinentes pour l'investigation. De plus ce projet fournit un moyen de «qualifier» les adresses en question. C'est à dire de pouvoir définir quel est le moyen d'envoi du courriel et/ou type d'envoi en question. En effet une adresse permet d'identifier une machine. Dès lors on peut essayer de savoir quel type de machine c'est. Il peut y avoir des machines zombies qui sont utilisées à distance pour envoyer l'e-mail. Ou encore des machines simples où l'utilisateur a directement envoyé le courriel via un client riche comme Outlook ou encore via un webmail tel que Hotmail. Le projet inclut le travail de Bachelor de Romain Cornu *Analyse de la source et du cheminement d'un e-mail*, ce qui permet d'ajouter la géo-localisation des adresses trouvées ainsi que la détection du moyen d'envoi (aussi appelé agent utilisateur, par ex : Outlook, Thunderbird, Hotmail, etc). Ce travail a aussi fourni un moyen d'extraire des adresses IP d'un en-tête. Ces différentes informations viennent s'ajouter à ce que fournit déjà ISCIA pour l'analyse de messages frauduleux et augmentera ainsi les capacités d'ISCIA et les possibilités de recoupement d'informations pour l'investigation. Ce projet devrait permettre une amélioration significative des analyses criminelles en donnant une capacité de définir l'origine d'un message frauduleux. Ce rapport est le rapport intermédiaire du projet.

Remerciements

Des remerciements tous particuliers sont adressés à M. Eduardo Sanchez, doyen du département TIC de la HEIG-VD qui m'a permis de faire ce travail. Je remercie également M. Christian Buchs, professeur responsable du projet, pour son soutien, sa disponibilité, sa compréhension et son suivi. Merci à M. Patrick Mast, mandant de ce projet, pour sa disponibilité, sa collaboration précieuse et son soutient dans ce projet. Finalement je remercie ma famille et mes amis pour leur soutient moral tout au long de ce travail.

Table des matières

1 Introduction **1**

1.1 Pré-étude . . . 2
1.1.1 En-tête de courriel . . . 3
1.1.2 Botnet . . . 6
1.1.3 whois . . . 7
1.1.4 Listes noires . . . 10
1.1.5 Structure du projet . . . 11
1.1.6 Performances . . . 13
1.2 Planification . . . 14
1.3 Spécifications . . . 17
1.3.1 API . . . 17
1.3.2 Journalisation . . . 18
1.3.3 Cas tests . . . 19

2 Analyseur modulaire **25**

2.1 Spécifications . . . 25
2.1.1 Contrôleur de modules . . . 26
2.1.2 Modules . . . 26
2.2 Conception . . . 26
2.2.1 Principes de fonctionnement de l'analyseur modulaire . . . 27
2.2.2 Principes de fonctionnement des modules . . . 28
2.2.3 Structure de l'analyseur modulaire . . . 30
2.2.4 Traitement des dépendances . . . 34
2.2.5 Fichier de configuration . . . 35
2.3 Réalisation . . . 36
2.3.1 Optimisation du code . . . 36
2.3.2 Méthode de traitement de requêtes sur les modules . . . 36
2.3.3 Journalisation . . . 36
2.3.4 Implémentation des énumérés de référence . . . 37
2.4 Tests unitaires . . . 37
2.4.1 Tests unitaires de «HeaderData» . . . 38
2.4.2 Tests unitaires de ModuleControler . . . 38
2.4.3 Test manuel . . . 38

3 Module IP Verification **41**
3.1 Analyse . . . 41
3.1.1 Analyse de l'extraction d'adresse de la machine Émettrice . . . 42
3.2 Conception . . . 43
3.2.1 Classe IPAnalyserModule . . . 44
3.2.2 Classe IPInfos . . . 44
3.2.3 Classe IPExctractor . . . 44
3.2.4 Classe IPChecker . . . 44
3.3 Réalisations . . . 45
3.4 Tests . . . 45

4 Module de géo-localisation **47**
4.1 Analyse . . . 47
4.2 Conception . . . 48
4.2.1 classe GeoLocModule . . . 48
4.2.2 classe GLDataChecker . . . 49
4.2.3 Classe UA_IPSource_IPLocIP . . . 50
4.2.4 Classe GeoLocation . . . 50
4.2.5 Fichier de configuration . . . 50
4.3 Réalisation . . . 51
4.4 Tests . . . 52

5 Module User Agent **53**
5.1 Analyse . . . 53
5.1.1 Classe UserAgentModule . . . 54
5.1.2 Classe UADataChecker . . . 55
5.1.3 Classe UA_IPSource_IPLocIP . . . 56
5.1.4 Classe UserAgent . . . 56
5.1.5 Fichiers de configuration . . . 56
5.2 Réalisation . . . 57
5.3 Tests . . . 57

6 Module Whois **59**
6.1 Spécification . . . 59
6.2 Conception . . . 60
6.2.1 Structure de base du module . . . 63
6.2.2 Gestion des données . . . 68
6.2.3 Fonctionnement du *thread* de mise à jour des données . . . 74
6.2.4 Fichier de configuration . . . 75
6.3 Réalisation . . . 75
6.3.1 Adaptation du code source trouvé . . . 75
6.4 Limitation du nombre de requêtes «whois» faites en cas de réponses nulle . . . 76
6.5 Tests . . . 77
6.5.1 Tests Unitaires . . . 77

7 Module IPBL **79**
7.1 Analyse . . . 79
7.2 Conception . . . 79
7.3 Réalisations . . . 80
7.4 Tests . . . 80

8 Validation **81**
8.1 Première validation . . . 81
8.2 Deuxième validation . . . 82

9 Retour sur le déroulement du projet **83**

10 Conclusion **85**
10.1 Retour sur les objectifs initiaux . . . 85
10.1.1 Performances . . . 86
10.2 Intégration . . . 89
10.3 Valeur ajoutée au projet ISCIA par EmailHeaderQualif . . . 89
10.4 Déroulement du projet . . . 90
10.5 Difficultés du projet . . . 90
10.6 Apport personnel . . . 90

A Résultats des tests de validation **103**
A.1 Résultats de l'étudiant . . . 103
A.2 Résultats du professeur responsables . . . 103
A.3 Résultats du mandants . . . 104

B Suivi du projet **105**

C Documentation de développement **107**
C.1 Utilisation des paramètres de l'API en entrée . . . 107
C.2 Structure des paquetages . . . 107
C.3 log4j . . . 108
C.3.1 Fichier POM pour la configuration de Maven . . . 108

D Manuel utilisateur du programme de test **109**
D.1 Installation de l'environnement . . . 109
D.1.1 Installation du JRE . . . 109
D.2 Manuel d'utilisation . . . 110
D.2.1 Lancement du programme de test et utilisation . . . 110
D.2.2 Fonctionnalités du programme de test . . . 111
D.2.3 Analyse des résultats et des informations affichées . . . 113
D.2.4 Modification de l'affichage des informations . . . 114
D.2.5 Erreurs connues pouvant survenir . . . 115

E DVD **117**

Chapitre 1

Introduction

L'abus de confiance est un moyen très vieux d'arnaquer les gens. La technique d'approche passe souvent par une mise en confiance de la personne pour la mettre dans une situation où il était possible de l'escroquer. Avec la démocratisation d'internet est apparu un nouveau moyen d'arnaquer les gens. Ce moyen est le courriel. De façon quasiment anonyme il est possible d'adresser un message à quelqu'un en le prenant à parti en lui décrivant, par exemple, une situation qui le touche. Si la cible de l'escroquerie a le malheur de répondre à cette sollicitation, il y a de fortes chances qu'elle se fasse arnaquer. Toute personne ayant une adresse de courriel qu'il utilise un temps soit peu, reçoit régulièrement des *spam*. Ce sont pour la plupart des messages publicitaires. Mais nombre d'entre eux sont aussi des tentatives d'arnaque. Pour n'en citer que deux, ce peut être du *phishing* (par ex. : lien sur un faux site de la poste où il faut entrer son identifiant et son mot de passe, qui sont ensuite récupérés pour une utilisation frauduleuse) ou une arnaque simple visant à payer des frais pour "aider" l'arnaqueur à se sortir d'une situation compliquée (aussi appelé *Nigerian Connection*[7]). Le montant de ces arnaques est estimé à 20 millions de dollars sous plainte selon [7] rien que pour les Pays-Bas. Et il faudrait, toujours selon la même source, le multiplier par 10 pour prendre en compte le montant réel de ces arnaques.

Le projet *ISCIA* (Internet Surveillance for Criminal Intelligence Analysis), développé au sein de le HES-SO, a entre autres pour but d'acquérir et d'analyser des pourriels pour investigation sur des fraudes. Dans cette analyse, *ISCIA* doit obtenir, si possible, l'adresse IP et diverses informations qui lui sont relatives. Le but de ce projet est donc de fournir les moyens pour obtenir ces informations. Il faut donc analyser des en-têtes de courriel pour pouvoir en extraire les adresses IP et donner des informations supplémentaires.

L'extraction d'adresses IP est un travail qui a déjà été fait par Romain Cornu dans son projet de Bachelor [14]. Ce travail comprend aussi la géo-localisation d'adresses IP et l'obtention du *user agent*. Ce travail sera donc intégré et adapté à ce projet. En plus des informations qui pourront être fournies par le travail de R. Cornu, ce projet a aussi pour but d'ajouter des informations telles que l'intervalle des adresses IP publiques et le propriétaire de cet intervalle ainsi que la qualification de ces adresses. Une qualification est une désignation permettant de dire «cette adresse est celle d'une machine zombie» ou encore «cette adresse est celle d'un proxy anonymiseur».

Finalement ce projet fournit donc une API qui prend en paramètre un en-tête de courriel et qui retourne toutes les informations qui peuvent en être extraites. De plus, de nouvelles fonctionnalités d'analyse doivent pouvoir facilement être rajoutées.

Les chapitres suivant traitent premièrement du système d'analyse (appelé analyseur modulaire) puis des différents modules qui le composent. Ensuite il y a un chapitre consacré à la validation du travail effectué et finalement se trouve un retour sur le déroulement du projet et la conclusion.

Dans la suite de ce chapitre se trouvent la pré-étude qui a permis d'établir le cahier des charges, une partie des spécifications et la structure du projet. Puis se trouve la planification du projet telle qu'elle a été décidée en fonction du cahier des charges.

1.1 Pré-étude

Une pré-étude de deux semaines a été faite pour établir le cahier des charges et les spécifications du projet. Elle a aussi permis de définir une structure au projet. Dans la suite de cette section se trouve les différentes réflexions et analyses en découlant.

La plus grande partie des deux semaines a été utilisée pour essayer de trouver les différents buts, moyens et raisons de qualification. Il a aussi fallu définir la liste des différentes qualifications possibles pour le cahier des charges et le moyen de qualifier des adresses IP.

Durant cette pré-étude ont été fait le listage des buts et des moyens dont dispose EmailHeaderQualif pour faire son travail mais aussi des réflexions sur son fonctionnement.

Le principal élément à voir est le cheminement d'un courriel. De l'envoyeur au destinataire il y a plusieurs points de passage. En partant du destinataire pour remonter à l'envoyeur il y a en premier le moyen de lecture du courriel. Celui-ci n'entrant pas dans les prérogatives de ce projet, il est inutile de s'étendre là-dessus. Avant la lecture du courriel il y a son cheminement. En faisant l'analogie à un courrier postal, peu importe que vous l'ayez reçu dans votre boîte au lettre ou retiré au guichet. Cela ne changera rien à son contenu ni aux informations sur l'envoyeur. En revanche les personnes traitant ce courrier peuvent vous apporter diverses informations. En remontant la filière, il y a en premier la personne qui vous remet le courrier. Ensuite il y a toutes les personnes qui ont eu votre courrier en main durant son transfert. Elles ne peuvent rien vous apprendre, sauf la première qui a retiré la lettre. Elle peut vous dire où la lettre a été retirée et parfois même qui la envoyée. Pour en revenir aux courriels, les personnes par qui passent le courriel sont les MTA (Mail Transfer Agent). Le premier MTA peut donc fournir des informations sur l'envoyeur. Or les MTA laissent une trace dans l'en-tête d'un courriel. Bien que certaines informations ne soient pas inscrites

Dans le cheminement du courrier, ce qui nous intéresserait dans notre cas ce sont les informations sur l'envoyeur. De pouvoir dire si c'est une personne, une victime d'un abus de confiance qui ne fait que renvoyer ce qu'on lui donne ou si il est impossible de définir qui

c'est car tout est anonyme dans le courrier. Définir quel est cet envoyeur et son but est la qualification.

Une personne qui envoie un courrier peut être connue comme étant un malfaiteur. Elle est donc qualifiée comme tel. Dans un courriel il est possible obtenir l'adresse de la machine ayant envoyé le courriel. C'est donc cette machine qui est qualifiée. Cependant cette machine peut être infectée par un maliciel qui permet de la contrôler à distance. Cette machine est alors appelé un zombie. C'est sa qualification. Une machine est donc identifiée par son adresse IP qui est elle-même qualifiée. Les machines zombies sont généralement utilisées en réseau. C'est-à-dire que plusieurs machines anonymes sont dirigées par un seul maître. Ce réseau est appelé un botnet et son étude est traitée un peu plus loin dans ce chapitre.

Toutes les informations citées précédemment se trouvent dans l'en-tête du courriel. C'est la partie cachée du courriel qui n'est pas visible par l'utilisateur. Il faut donc extraire, si possible, les adresses IP de l'envoyeur et du MTA ainsi que les différentes informations de cet en-tête. L'étude sur les en-tête se trouve elle aussi à la suite de ce chapitre.

Finalement, il faut pouvoir obtenir différentes informations sur les adresses IP extraites. Il a été trouvé deux moyens d'obtenir des informations en plus de celle déjà fournies par le travail de Romain Cornu. Le premier est une commande bien connue nommée «whois» (voir [6]) et qui permet de consulter les bases de données des organismes gérant les adresses IP et les informations qui y sont relatives. Le deuxième est un système de listes noires utilisé pour filtrer les spams venant de machines listées comme étant des envoyeurs connus de spam.

Dans la suite de cette section se trouvent les différentes études faites. La première concerne les en-têtes de courriel. La seconde concerne les botnets, la troisième le whois, et la quatrième les listes noires. Finalement se trouve l'étude sur la structure du projet.

1.1.1 En-tête de courriel

Les en-têtes de courriel contiennent des informations sur son cheminement de l'envoyeur au destinataire. Les normes utilisées pour construire cet en-tête se trouvent dans [24]. Ici sont traités leur structure et leur interprétation.

La première chose à voir dans les en-têtes ou *header*, c'est qu'ils sont séparés du reste du message par un double *CRLF*. Un CRLF est un marqueur composé de deux caractères ASCII 13(0x0D) et 10(0x0A) qui désigne une fin de ligne.

Tout ce qui vient après n'est que le contenu du message et ne nous intéresse pas dans le cadre de ce travail.

Tous les champs contenus dans un en-tête sont sous la forme «clé :valeur». Un champ se termine avec le caractère de fin de ligne (CRLF), comme dans l'exemple ci-dessous :

```
Cle1:Valeur1 CRLF
Cle2:Valeur2 CRLF
Cle3:Valeur3 CRLF
Cle4:Valeur4 CRLF
CleX:ValeurX CRLF
CRLF
CRLF
corps du message
```

Dans l'en-tête se trouvent potentiellement toutes les informations sur le cheminement d'un courriel et sur sa source. Ci-dessous se trouve un exemple d'en-tête.

```
Delivered-To: xxxxxxx@gmail.com
Received: by 10.100.247.17 with SMTP id u17cs54126anh;
        Wed, 20 Oct 2010 09:11:04 -0700 (PDT)
Received: by 10.150.191.12 with SMTP id o12mr269888ybf
    .209.1287591063755;
        Wed, 20 Oct 2010 09:11:03 -0700 (PDT)
Received-SPF: softfail (google.com: best guess record for domain
    of transitioning Hetty.Versnel@INHOLLAND.nl does not designate
     194.171.35.18 as permitted sender) client-ip=194.171.35.18;
Received: by 10.243.50.136 with POP3 id 8mf146158gyg.27;
        Wed, 20 Oct 2010 09:11:03 -0700 (PDT)
X-Gmail-Fetch-Info: xxxxx@hotmail.com 1 pop3.live.com 995
    xxxxxx@hotmail.com
X-Message-Delivery: Vj0xLjE7dXM9MDtsPTE7YT0xO0Q9MTtTQ0w9MA==
X-Message-Status: n
X-SID-PRA: Versnel, Hetty <Hetty.Versnel@INHOLLAND.nl>
X-AUTH-Result: NONE
X-Message-Info: PmtKMWZUblr6ef2wSNeO+is/
    sST8y9Ka9j3420DYUrvNGTTGis3zcq8EhSmLhLp/SZ1a7hzQewPjRYq+PZqjL/
    zoJn9qabox6INhGCvfMAE=
Received: from smtp3.inholland.nl ([194.171.35.18]) by col0-mc3-
    f18.Col0.hotmail.com with Microsoft SMTPSVC(6.0.3790.4675);
         Wed, 20 Oct 2010 08:16:03 -0700
Received: from (unknown [194.171.35.190]) by smtp3.inholland.nl
    with smtp
         id 079c_83d0_fe98980c_dc5b_11df_b794_001422241924;
        Wed, 20 Oct 2010 17:09:22 +0200
Received: from SR0482.intranet.inholland.nl (10.1.67.93) by
 SR0006.intranet.inholland.nl (10.1.70.72) with Microsoft SMTP
     Server (TLS) id
 8.2.254.0; Wed, 20 Oct 2010 17:08:57 +0200
Received: from SR0478.intranet.inholland.nl ([fe80::9023:a0a0:
    f840:9180]) by
 SR0482.intranet.inholland.nl ([fe80::50d7:d7a9:9ba6:682a%14])
     with mapi; Wed,
```

```
 20 Oct 2010 17:08:56 +0200
From: "Versnel, Hetty" <Hetty.Versnel@INHOLLAND.nl>
Date: Wed, 20 Oct 2010 17:08:55 +0200
Subject: Confirmation Email Ref No: (JSPSD-11532264832140)
Thread-Topic: Confirmation Email Ref No: (JSPSD-11532264832140)
Thread-Index: AQHLcGi3M8ZZunKuz0mzQzT8SE90+Q==
Message-ID: <7304E1C44E5FD14A97761A2A26D55B9C0464912F47@SR0478.
    intranet.inholland.nl>
Accept-Language: nl-NL
Content-Language: nl-NL
X-MS-Has-Attach:
X-MS-TNEF-Correlator:
acceptlanguage: nl-NL
Content-Type: text/plain; charset="iso-8859-1"
Content-Transfer-Encoding: quoted-printable
MIME-Version: 1.0
To: Undisclosed recipients:;
Return-Path: Hetty.Versnel@INHOLLAND.nl
X-NAI-Spam-Flag: NO
X-NAI-Spam-Level: ***
X-NAI-Spam-Threshold: 5
X-NAI-Spam-Score: 3
X-NAI-Spam-Version: 2.2.0.9286 : core <3654> : streams <556394> :
     uri <70333
X-OriginalArrivalTime: 20 Oct 2010 15:16:06.0541 (UTC) FILETIME=[
    B7A98BD0:01CB7069]
```

On peut y voire les différents champs de type «clé :valeur» tel que «Date : Wed, 20 Oct 2010 17 :08 :55 +0200» mais surtout on y trouve les champs «Received».

La principale caractéristique qui nous intéresse dans notre cas c'est la liste des champs *received*. Ces champs sont introduits par les MTA lorsqu'ils reçoivent le courriel avant de le transmettre. Normalement se trouvent dans cet en-tête l'adresse IP de la source de cet e-mail et l'adresse IP du premier MTA ayant fait le transfert du courriel. Le dernier *received* inséré dans l'en-tête (le plus haut dans un fichier) est celui inséré par le dernier MTA. Le premier *received* inséré peut donc être celui du premier MTA et contenir les informations nécessaires pour pouvoir en extraire les adresses IP désirées. Le deuxième peut aussi être intéressant pour pouvoir confirmer l'adresse du premier MTA ou l'obtenir au cas où elle ne serait pas renseignée dans le premier. Il est important de savoir que la norme concernant les received n'est pas toujours respectée et que parfois il n'est pas complétement renseigné comme on peut le voir dans l'en-tête d'exemple précédent. Le standard ne précise pas exactement quelle adresse IP il faut placer dans les différents champs ce qui laisse la possibilité aux différents MTA de mettre plusieurs adresses différentes. Par exemple, le champ «from» peut contenir l'adresses IP de la machine émettrice du courriel ou celle du MTA. Ce qui fait que les adresses IP extraites ne sont pas fiables à 100%.

Cependant il faut aussi être conscient de la pertinence de ces informations. En effet, un

MTA ne fait que reprendre telles-quelles les informations contenues dans le courriel et rajoutant au dessus ses propres champs. Ainsi, il n'y a aucun contrôle sur ce qui a été inséré avant. Il est donc possible qu'une partie de l'en-tête soit forgé. C'est-à-dire que des éléments peuvent avoir été insérés par l'envoyeur du message pour brouiller les pistes et empêcher de remonter jusqu'à lui.

En plus de cela, il est possible d'anonymiser l'envoi des courriels. Pour ce faire il suffit de passer par un proxy anonymiseur. Les adresses IP qui seront obtenue sur l'extraction seront donc celles du proxy et non de l'envoyeur. Ces adresses peuvent aussi faire l'objet d'une qualification.

L'extraction d'adresses IP peut se faire sur plusieurs types d'adresses, les adresses IPV4 et IPV6 (voir [1]). Seules les adresses IPV4 seront utilisées ici car le protocole IPV6 n'est encore que très peu utilisé.

L'extraction d'adresses IP est détaillée dans le chapitre 3

1.1.2 Botnet

Dans cette section se trouve les différentes recherches effectuées sur les botnets.

Comme dit précédemment un botnet est un réseau d'ordinateurs «zombies» qui obéissent à un maître. Il peuvent servir à envoyer des courriels depuis l'ordinateur d'une personne qui l'ignore totalement. Le vrai envoyeur reste donc anonyme et protégé de toutes recherches. En ajoutant à cela la possibilité de forger les en-têtes, il devient très difficile de trouver le réel envoyeur du courriel. La figure ci-dessous présente la liste des différents botnets les plus utilisés pour envoyer des messages de spam. Ces statistiques sont basées sur des «honeypots» (boîte de courriel servant d'appât à des spams).

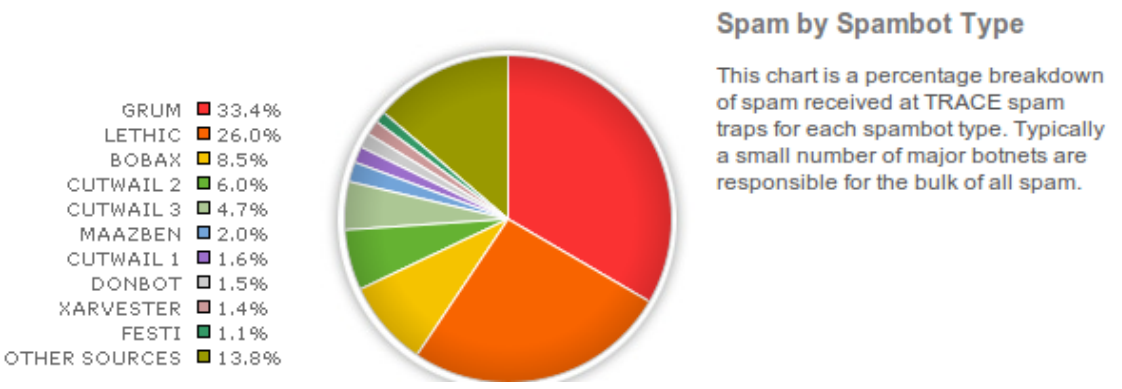

Spam by Spambot Type
This chart is a percentage breakdown of spam received at TRACE spam traps for each spambot type. Typically a small number of major botnets are responsible for the bulk of all spam.
GRUM 33.4%
LETHIC 26.0%
BOBAX 8.5%
CUTWAIL 2 6.0%
CUTWAIL 3 4.7%
MAAZBEN 2.0%
CUTWAIL 1 1.6%
DONBOT 1.5%
XARVESTER 1.4%
FESTI 1.1%
OTHER SOURCES 13.8%

FIGURE 1.1 – Répartition des différents botnets (source [10]) octobre 2010

L'idée de base de recherche dans ce projet était de faire des tests d'envoi de courriels via un

bot et d'analyser ces courriels pour en faire ressortir des règles de détection. Cependant la difficulté de trouver un logiciel de botnet qui fonctionne et dont les sources sont complètes et disponibles a posé un gros problème et une grosse perte de temps. En effet, la plupart des botnet fonctionne via IRC. C'est-à-dire que le client sur l'ordinateur infecté se connecte à un canal IRC sur un serveur prédéfini pour recevoir ses ordres. Cependant, les informations de connexion se trouvent dans le code du client. Ce qui implique de re-compiler le client pour pouvoir créer un botnet. L'obtention de sources fut déjà difficile. Rbot [34] et Hexbot ont été trouvés. Cependant ces bots n'étaient pas fait pour envoyer des courriels. La recherche et les tests de botnet pour trouver des règles de détection de botnets dans un en-tête se sont donc trouvées infructueuses.

Cependant la recherche sur les botnets n'a pas été totalement vaine. En effet, des listes noires d'adresses IP existent. À l'origine elles sont faites pour contrer les spams. C'est-à-dire que les analyseurs de spam peuvent se baser sur ces listes pour filtrer les messages reçus et détecter les spams. Ces listes d'adresses sont utilisées aussi pour d'autres usages. Ceci est traité dans la section 1.1.4

Des recherches supplémentaires ont été faites en bibliothèque pour essayer de trouver des sources d'informations sur les botnets. Cependant aucune publication scientifique sur le sujet n'a été trouvé.

Le chapitre suivant traite des différentes informations qu'une commande «whois» retourne.

1.1.3 whois

La commande whois permet d'obtenir des informations sur une adresse IP ou un nom DNS. Ci-après se trouve un exemple de résultat d'exécution de commande whois sur l'adresse IP 11.22.33.44 :

```
#
# Query terms are ambiguous.  The query is assumed to be:
#     "n 11.22.33.44"
#
# Use "?" to get help.
#


#
# The following results may also be obtained via:language=Octave,
                    % choose the language of the code
basicstyle=\footnotesize,       % the size of the fonts that are
    used for the code
numbers=left,                   % where to put the line-numbers
numberstyle=\footnotesize,      % the size of the fonts that are
    used for the line-numbers
stepnumber=2,                   % the step between two line-
    numbers. If it's 1 each line
```

```
                                % will be numbered
numbersep=5pt,                  % how far the line-numbers are
    from the code
backgroundcolor=\color{white},  % choose the background color.
    You must add \usepackage{color}
showspaces=false,               % show spaces adding particular
    underscores
showstringspaces=false,         % underline spaces within strings
showtabs=false,                 % show tabs within strings adding
     particular underscores
frame=single,                   % adds a frame around the code
tabsize=2,                      % sets default tabsize to 2
    spaces
captionpos=b,                   % sets the caption-position to
    bottom
breaklines=true,                % sets automatic line breaking
breakatwhitespace=false,        % sets if automatic breaks should
     only happen at whitespace
title=\lstname,                 % show the filename of files
    included with \lstinputlisting;
                                % also try caption instead of
                                    title
escapeinside={\%*}{*)},         % if you want to add a comment
    within your code
%morekeywords={*,...}
# http://whois.arin.net/rest/nets;q=11.22.33.44?showDetails=true&
    showARIN=false
#

NetRange:       11.0.0.0 - 11.255.255.255
CIDR:           11.0.0.0/8
OriginAS:
NetName:        DODIIS
NetHandle:      NET-11-0-0-0-1
Parent:
NetType:        Direct Allocation
RegDate:        1984-01-19
Updated:        2007-08-22
Ref:            http://whois.arin.net/rest/net/NET-11-0-0-0-1

OrgName:        DoD Network Information Center
OrgId:          DNIC
Address:        3990 E. Broad Street
City:           Columbus
StateProv:      OH
PostalCode:     43218
```

```
Country:            US
RegDate:
Updated:            2010-08-25
Ref:                http://whois.arin.net/rest/org/DNIC

OrgTechHandle:  MIL-HSTMST-ARIN
OrgTechName:    Network DoD
OrgTechPhone:   +1-614-692-2708
OrgTechEmail:   HOSTMASTER@nic.mil
OrgTechRef:     http://whois.arin.net/rest/poc/MIL-HSTMST-ARIN

OrgTechHandle:  REGIS10-ARIN
OrgTechName:    Registration
OrgTechPhone:   +1-800-365-3642
OrgTechEmail:   REGISTRA@nic.mil
OrgTechRef:     http://whois.arin.net/rest/poc/REGIS10-ARIN

#
# ARIN WHOIS data and services are subject to the Terms of Use
# available at: https://www.arin.net/whois_tou.html
```

Les lignes précédées d'un «#» sont des commentaires ajoutés automatiquement par le serveur qui réponds à la requête. Le restant est sous la forme «clé :valeur» et donne les informations disponibles dans la base de données du serveur répondant à la requête.
Il s'y trouve une liste d'informations intéressantes disponibles, décrites ci-après :

NetRange Intervalle d'adresses dans laquelle se trouve l'adresse IP

NetName Nom de l'organisation à laquelle appartient cette adresse. Le champ **OrgName** ou Org-Name fournit aussi cette information et souvent de manière plus explicite.

Le restant des informations varie selon la base de données utilisée pour répondre à la requête «whois». En effet, plusieurs organismes sont chargés de gérer les adresses IP. C'est à ces organismes que la requête est adressée selon le type d'adresse. Dans l'exemple ci-dessu c'est ARIN (voir [25]) qui réponds à la requête. Si la requête est faite sur une adresse IP européenne tel que 178.198.134.179, ce sera RIPE (voir [15]) qui va répondre, l'organisme de gestion européen. Ces organismes se nomme RIR (Registre Internet Régional)[35] et sont : ARIN, RIPE, APNIC, LACNIC et AfriNIC.

Chacun a une façon différente de répondre. La notion de clé/valeur reste pour chacun des organismes, mais les clés changent entre eux. De plus, les autres informations ne semblent pas directement pertinentes pour une investigation.

Une liste de serveur whois est disponible sur [13].

Il en résulte une partie à développer exploitant les informations qu'un whois peut retourner ou un équivalant.

La prochaine section traite des listes noires et de ce qu'elles peuvent apporter au projet.

1.1.4 Listes noires

Cette partie traite des listes noires d'adresses IP. Celles-ci sont utilisées à l'origine pour filtrer les courriels. Par exemple [11] permet de le faire.

Pour tester si une adresse IP fait partie d'une liste noire, la commande «nslookup»[12] est utilisée. Cette commande est à l'origine pour résoudre un nom DNS en adresse IP. Cependant, en utilisant correctement cette commande, il est possible de faire une requête sur un serveur de liste noire. Il suffit de concaténer l'adresse IP avec l'adresse DNS du serveur de liste noire. Ceci revient au même que configurer le serveur DNS de la machine par celui du serveur de liste noire. Par exemple pour faire une requête sur la liste «zen» de *SpamHaus* pour l'adresse IP, il suffit d'écrire la commande « *nslookup 2.0.0.127.zen.spamhaus.org*»

La réponse à cette requête est la suivante :

```
Non-authoritative answer:
Name:    2.0.0.127.zen.spamhaus.org
Address: 127.0.0.10
Name:    2.0.0.127.zen.spamhaus.org
Address: 127.0.0.4
Name:    2.0.0.127.zen.spamhaus.org
Address: 127.0.0.2
```

Il en ressort une série d'adresses 127.0.0.x retournées. Vu qu'au moins une adresse est retournée, cela signifie que l'adresse pour laquelle il a été fait une requête est dans la liste noire. Selon le serveur de liste noire utilisé on peut en déduire une qualification. Le «x» est un numéro permettant d'interpréter le résultat. Par exemple un 1 peut signifier «Peu fiable» ou encore «Proxy anonymiseur». Ce résultat peut donc aussi être utilisé pour qualifier une adresse IP. Au contraire la réponse pour l'adresse 11.22.33.44 sera :

```
** server can't find 11.22.33.44.zen.spamhaus.org: NXDOMAIN
```

Le NXDOMAIN signifie que l'adresse n'est pas dans la liste noire.

Les listes noires sont appelées DNSBL. Le site [26] référence toute une série de listes noires pour le développement de la partie de qualification par listes noires. Chacun de ces DNSBL n'a pas forcément la même signification. Par exemple `xbl.spamhaus.org` référence des IP connues comme étant infectées (machines zombie par exemple). Alors que `b.barracudacentral.org` donne une réputation de l'adresse IP.

La section suivante traite de la structure du projet à mettre en place pour pouvoir faire les différentes analyses précédemment citées.

1.1.5 Structure du projet

Cette section présent la réflexion sur la structure et le fonctionnement du projet.

La première partie de cette réflexion concerne la façon dont va être utilisé le projet. En effet, le projet doit s'intégrer à *ISCIA* pour fournir des informations sur les en-têtes. Le plus simple étant dans notre cas d'interfacer *ISCIA* avec le système d'obtention d'informations. C'est pourquoi une API a été choisie. En plus, une API présente l'avantage de pouvoir être externalisée d'*ISCIA* et peut donc, par exemple, plus facilement tourner sur un autre serveur. Mais encore, cela permet de l'utiliser facilement dans un autre projet que *ISCIA*

La deuxième partie a été de trouver un nom à cette API. Le nom choisi est EmailHeader-Qualif . Il met en avant que le but premier de cette API est de qualifier des adresses IP contenues dans un en-tête de courriel.

La troisième partie a été de définir le fonctionnement de cet API. Précédemment a été montré un en-tête de courriel. Au premier coup d'œil il apparaît que la quantité d'informations contenue dans un en-tête est très grande et très variée. EmailHeaderQualif n'ayant pas la prétention de couvrir complètement une analyse d'en-tête, il a fallu prendre en considération qu'il devait être facile d'agrandir le projet. C'est pourquoi une structure modulaire a d'entrée été prise en compte.

L'idée derrière une structure modulaire est de pouvoir gérer différents modules qui extraient et fournissent des informations sur un en-tête. Cela implique qu'au centre du projet se trouve un «cœur» qui gère les différents modules. Il est possible de voir dans la figure 1.2 ci-dessous ce gestionnaire de module qui s'appelle «Module Controler». L'ensemble du projet doit donc se composer de deux parties. La partie visible de l'iceberg qui est l'API EmailHeaderQualif et la partie cachée qui englobe les modules et le gestionnaire de module nommé «Analyseur modulaire».

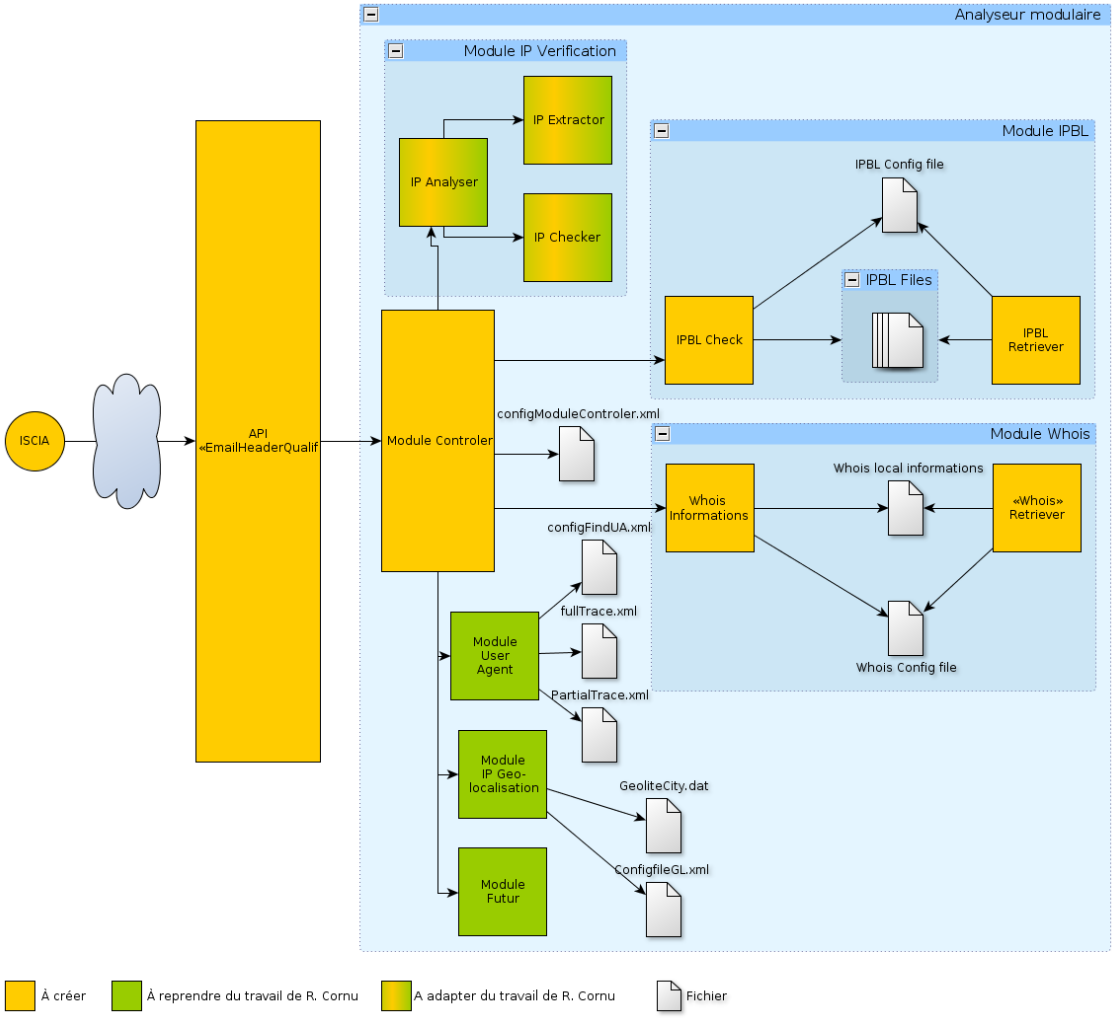


FIGURE 1.2 – Vue globale du projet

Comme dit précédemment, le travail de Romain Cornu [14] fournit une extraction d'adresses IP depuis un en-tête, de la géo-localisation d'adresses IP et l'obtention du *user agent* depuis un en-tête. Ces différentes parties complètent ce projet et doivent donc s'intégrer à celui-ci. Chacune d'entre elle est intégrée au projet comme un module. Ce qui nous donne donc trois modules à ajouter à l'analyseur modulaire :

IP Verification Ce module se base sur l'extraction fournie dans [14]. Cependant il doit être adapté car il ne détecte pas systématiquement toutes les adresses IP requises (il ne prend que l'adresse IP de la source et seulement dans certains cas de figures).

User Agent La partie *user agent* du travail de R. Cornu est à intégrer telle-quelle dans le projet comme un module.

IP Geo-localisation La géo-localisation est à intégrer telle-quelle dans le projet en tant que module.

De la réflexion sur le whois (1.1.3) et sur les listes noires (1.1.4) découlent les modules «IPBL» pour IP Black Listing qui permet de qualifier les adresses IP et «Whois» qui permet d'obtenir l'intervalle des adresses IP et le propriétaire de cette adresse.

Un autre aspect de EmailHeaderQualif est les fonctions qu'elle fournit et les paramètres de ces fonctions. Au premier abord il apparaît qu'une seule fonction soit nécessaire. Elle doit prendre en paramètre un en-tête et retourner les différentes informations trouvées par l'analyseur modulaire. Cependant en s'inspirant de [14], il apparaît que les fonctions fournies prennent en paramètre un en-tête et la liste des *received* contenus dans cet en-tête. En effet, *ISCIA* fait déjà de l'extraction d'en-tête. Il n'est donc pas nécessaire de refaire le même travail. Il suffit de rajouter la liste de ces en-têtes dans les paramètres de la fonction fournie par EmailHeaderQualif .

Le nom de la fonction devrait idéalement être le même que celui de l'API. Cependant pour des raisons de codage c'est impossible. Le nom qui lui a été donné est headerQualif .

Ce que retourne la fonction est définie dans le cahier des charges. Le retour de la fonction se fait via une table de hachage contenant les différents champs. Suite à la décision de M. C. Buchs, professeur responsable, c'est un objet de ce type qui est utilisé afin de réduire le couplage entre *ISCIA* et EmailHeaderQualif . En effet, créer un objet spécifique pour le retour de headerQualif augmenterait le couplage en cas de modification ultérieur de l'objet pour adaptation ou amélioration du projet.

Les réflexions précédentes, l'analyse du fonctionnement du travail de R. Cornu et les discussions avec le mandant aboutissent à une fonction avec la signature suivante donnée dans le cahier des charges :

Map<String, String> headerQualif (header Map<Srtring><String>, receiveds List<Map<String><String»)

La suite traite des performances devant être fournie par le projet et les conséquences que cela a.

1.1.6 Performances

L'évaluation des performances se base sur les besoins d'*ISCIA*. En collaboration avec le mandant, il a été établi que l'analyseur modulaire devait pouvoir analyser environ 100'000 en-têtes en 24 heures soit un en-tête toute les 0.86 secondes. Ce qui implique qu'aucun accès à une ressource extérieure ne doit ralentir le traitement. Ces accès peuvent prendre la forme d'accès disque ou de requêtes sur un autre serveur.

Les conséquences sont que toutes les données utilisées par les modules doivent être chargées en mémoire pour réduire au maximum ces temps.

Durant cette pré-étude, le cahier des charges se trouvant au début du document a été fait, ainsi que la planification qui suit, correspondant à ce cahier des charges et à la pré-étude.

1.2 Planification

Ici sont indiqués les différentes étapes ayant découlé du cahier des charges établi et de la pré-étude. Le découpage des tâches se base sur la structure modulaire. En effet, la principale et première tâche à accomplir est la base de la structure et donc le gestionnaire de modules. Puis vient le module d'extraction d'IP qui permet d'obtenir les adresses nécessaires pour certains des autres modules. Il s'ensuit la géo-localisation et le module d'obtention du *user agent* qui sont supposés rapides à intégrer. Puis le module «whois» car suite aux différents essais faits durant la pré-étude, il semblerait aisé d'adapter du code java trouvé pour faire cette tâche. Et finalement, pour le développement, le module IPBL qui est placé en dernier car c'est le module le plus "à risques". En effet, l'obtention des sources de données pour avoir des données locales risque de poser problème. Un des moyen d'obtenir une informations sur une adresse IP semble être par une requête *nslookup* et ce n'est pas tolérable (voir 1.1.6). Un autre moyen serait d'obtenir une liste d'adresses (base de données ou simple liste). Mais il est difficile de trouver de telles sources de données. Les filtres anti-spam utilisent ce genre de listes.

Le diagramme 1.3 sur la page suivante présente chaque phase de développement, détaillée ci-après, comportant une spécification, une conception, une série de tests et des corrections.

Analyseur modulaire C'est la base du projet et la partie qui prend le plus de temps et qui comprend les parties suivantes

[Codage structure] Codage de la structure de l'analyseur. Ceci comprend le contrôleur de modules et la structure objet pour la gestion des modules.

[Module IP Verification] Le développement de ce module est pris dans cette phase de projet. Le but étant d'avoir une base d'analyse pour la validation. Après cette tâche, il sera possible d'extraire des adresses IP d'un en-tête de courriel.

[Intégration du module User Agent] Ce module est aussi inclus dans l'analyseur car son développement est supposé court et pourra être inscrit durant la première validation. Après cette tâche il sera possible d'obtenir le user-agent à partir d'un en-tête de courriel.

[Intégration du module Géo-Localisation] Ce module est aussi inclus dans l'analyseur car son développement est supposé court et pourra être inscrit durant la première validation. Après cette tâche les IP publiques pourront être géo-localisées.

Module informations IP (whois) Ce module est séparé de l'analyseur modulaire car le travail à fournir pour son développement est conséquent. De plus les «risques» temporels qu'il représente sont plus grands car aucun code source permettant de faire des requêtes whois n'a été trouvé pour l'instant. Après cette tâche, il sera possible d'avoir des informations supplémentaires sur les adresses IP publiques telles que l'intervalle d'adresses comprenant cette adresse.

Module listes noires IP (IPBL) Tout comme le module *whois*, le développement de ce module est conséquent. De plus cette partie est «si le temps le permet». Elle est donc séparée de l'analyseur modulaire. À la fin de cette partie, il sera possible de qualifier les adresses IP extraites.

Rapport Cette tâche comprend plusieurs sous-tâches pour chacune des phases du projet. Le but est la rédaction du rapport. Cependant il faut prendre en compte que toute la partie de spécification et de conception inclut aussi une partie de rapport.

Validation La validation est faite par le mandant et le professeur responsable. Le temps de travail pris par cette validation n'est donc pas comptabilisé dans le temps de travail. Cependant le temps des corrections à apporter suite à ces validations est pris en compte.

Suivi Cette tâche est la partie de gestion du projet. Elle prend en compte la rédaction des documents de suivi (journal de travail et rapport hebdomadaire) et les entretiens hebdomadaires avec le professeur responsable.

Le temps de codage de chacune des parties est une estimation personnelle. En revanche elle sert de base pour calculer les temps de spécification, conception, test et correction. En effet, le calcul se base sur un ratio personnel :

- 40% du temps de la tâche est pris par le codage
- 10% du temps de la tâche est pris pour les spécifications
- 20% de la tâche pour la conception
- 10% de la tâche pour les tests unitaires
- 20% pour la correction

Comme on peut le voir dans la figure 1.3 les tâches de suivi ne sont pas prises en compte dans le déroulement du projet. C'est parce-qu'elles sont réparties sur les différentes semaines du projet.

Le prochain chapitre donne les spécifications découlant de la pré-étude et du cahier des charges.

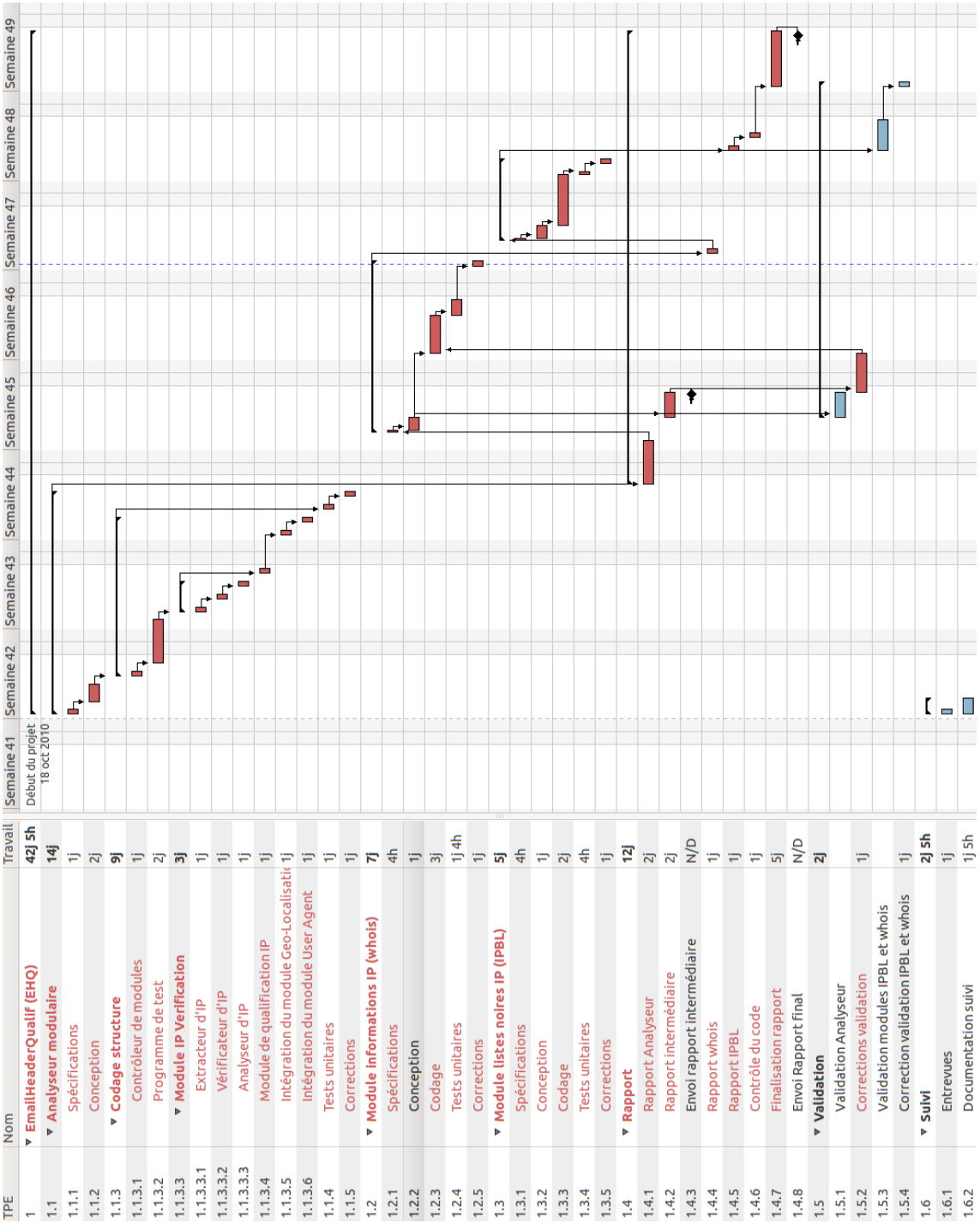


FIGURE 1.3 – Diagramme de Gant de la Planification

1.3 Spécifications

Dans cette partie sont données les spécifications découlant de la pré-étude et du cahier des charges. Puis sont donnés les cas-test correspondants aux spécifications.

Ces spécifications sont générales au projet. Elles concernent EmailHeaderQualif "vu de l'extérieur". Ce sont celles fournies pour le mandant et le professeur responsable et validées par leurs soins.

1.3.1 API

Cette section liste les spécifications du projet pour l'API

N°	Explication
1.3.1.1	EmailHeaderQualif fournit une méthode de qualification (c.f. Cahier des charges) d'en-têtes nommée headerQualif
1.3.1.2	headerQualif prend en premier paramètre une table de hachage contenant l'en-tête complet
1.3.1.3	headerQualif prend en deuxième paramètre la liste des received. Chacun d'eux est donné sous forme d'une table de hachage
1.3.1.4	Le retour de headerQualif se fait sous forme d'une table de hachage
1.3.1.5	Les noms des clés dans la table de hachage du retour de headerQualif correspondent à ceux donnés dans le cahier des charges (c.f. Cahier des charges)
1.3.1.6	Si une information n'a pas pu être trouvée, l'entrée dans la table de hachage n'existe pas
1.3.1.7	headerQualif retourne, si possible, le user-agent
1.3.1.8	headerQualif retourne, si possible, l'adresse IP privée de la machine ayant envoyé le courriel
1.3.1.9	headerQualif retourne, si possible, l'adresse IP publique du premier MTA [1]
1.3.1.10	headerQualif qualifie (c.f. Cahier des charges), si possible, l'adresse IP du premier MTA
1.3.1.11	headerQualif retourne, si possible, l'adresse IP publique de la machine ayant envoyé le courriel
1.3.1.12	headerQualif qualifie (c.f. Cahier des charges), si possible, l'adresse IP publique de la machine ayant envoyé le courriel
1.3.1.13	headerQualif retourne, si possible, l'intervalle de l'adresse IP publique de la machine ayant envoyé le courriel
1.3.1.14	headerQualif retourne, si possible, le nom de l'organisation à laquelle appartient l'intervalle d'adresses IPs publiques de l'envoyeur
1.3.1.15	headerQualif retourne, si possible, les informations de géo-localisation [2] de l'adresse IP publique de la machine ayant envoyé le courriel

1. On parle ici de l'adresse IP publique de la machine ayant introduit le premier received
2. c.f. Cahier des charges pour le détail des informations fournies

N°	Explication
1.3.1.16	Les qualifications retournées pour une même adresse IP sont dans des champs différents. La clé d'accès à ces champs est différenciée par un numéro de 1 à x où x est le nombre total de qualifications pour une IP (c.f. liste des valeurs retournées dans le cahier des charges)
1.3.1.17	Si aucune information pertinente n'a pu être trouvée lors de l'appel de la méthode headerQualif , son retour est *null*
1.3.1.18	Si l'en-tête fourni à headerQualif est inconsistant (nulle ou de mauvais type), lever une exception
1.3.1.19	EmailHeaderQualif doit pouvoir traiter 100'000 en-têtes par jour soit un en-tête toutes les 0.86 secondes
1.3.1.20	EmailHeaderQualif n'a pas besoin de prendre en charge des appels concurrents. C'est à dire que ISCIA ne fera jamais deux appels simultanés de la méthode headerQualif
1.3.1.21	Seules les erreurs du standard Java™ sont utilisées (pas de redéfinition d'erreurs)
1.3.1.22	Les données locales[3] nécessaires sont chargées en mémoire lors du lancement de EmailHeaderQualif
1.3.1.23	Si les données locales sont indisponibles, EmailHeaderQualif ne démarre pas
1.3.1.24	EmailHeaderQualif met à jour périodiquement les données en mémoire à partir des données locales sans ralentir le processus de qualification
1.3.1.25	L'intervalle de mise à jour des données en mémoire est définie dans les fichiers de configuration des différents modules
1.3.1.26	Le traitement peut monopoliser le processeur durant son exécution mais ne doit pas bloquer la machine (pas de temps réel ou de priorité haute)
1.3.1.27	La seule partie visible de EmailHeaderQualif est la méthode headerQualif et les méthodes d'initialisation et de terminaison de EmailHeaderQualif . Tout le reste ne doit être accessible que en interne à EmailHeaderQualif

1.3.2 Journalisation

Cette section liste les spécifications pour la journalisation.

N°	Explication
1.3.2.1	La journalisation se fait avec log4j
1.3.2.2	Tout événement levant une exception doit être journalisé
1.3.2.3	Quand un système de mise à jour des données locales est démarré, une notification doit apparaître dans le journal au début et à la fin de la mise à jour avec l'état de la mise à jour (réussite/échec)
1.3.2.4	La mise à jour périodique des données en mémoire de EmailHeaderQualif depuis les données locales doit être journalisée ainsi que son résultat (réussite/échec)

3. On entend par données locales, les données stockées dans un fichier et utilisées par EmailHeaderQualif

N°	Explication
1.3.2.5	Le démarrage et l'arrêt de EmailHeaderQualif doivent être journalisés
1.3.2.6	Tout ce qui n'est pas précisé précédemment n'est pas journalisé

1.3.3 Cas tests

Le fichier de log auquel on se réfert dans les cas-tests s'appelle «EHQlog.log» et se trouve dans le répertoire d'exécution du programme. Il est conseillé de lire le manuel d'utilisation du programme de test avant d'effectuer ces cas-tests. Ces cas-test ont été élaborés pour couvrir les spécifications et le cahier des charges. Ils ont aussi été élaborés à partir des tests faits dans [14].

N° cas test	N° spécification	Description	Résultat attendu
1.3.3.1	1.3.1.5	Lancer le programme de test et choisir l'option «afficher les retours possibles»	La liste fournie correspond précisément à la liste définie dans le cahier des charges : UserAgent PrivateIP PublicIP PublicIPQualification PublicIPNetName PublicIPRange PublicIPcountryCode PublicIPcountryName PublicIPzipcode PublicIPcityName PublicIPlongitude PublicIPlatitude PublicIPmetroCode PublicIPaeraCode PublicIPdmaCode PublicIPtimeZone FirstMTAIP FirstMTAIPQualification PublicIPQualification ne comporte pas de numéro. C'est lors de l'ajout d'une qualification que le numéro est ajouté automatiquement. Il en va de même pour FirstMTAIPQualification.

N° cas test	N° spécification	Description	Résultat attendu
1.3.3.2	1.3.1.7	Lancer une analyse sur les fichiers dans le dossier test_User_Agent	L'analyse de chacun des fichiers doit sortir un *user agent* sauf l'analyse de UserAgent_no_info.txt qui ne retourne rien. on doit trouver
		UserAgent_P_Mail-System-Version.txt	UserAgent = Outlook
		UserAgent_P_X-Newsreader_OE.txt	UserAgent = null
		UserAgent_F_Mailer.txt	UserAgent = Mozilla Thunderbird
		UserAgent_A_Message-ID.txt	UserAgent = AOL Mail
		UserAgent_P_Mailer.txt	UserAgent = Outlook
		UserAgent_P_X-Newsreader.txt	UserAgent = Outlook
		UserAgent_F_Mail-System-Version.txt	UserAgent = Mozilla Thunderbird
		UserAgent_Y_Dkim-Signature.txt	UserAgent = Yahoo Mail
		UserAgent_A_Dkim-Signature.txt	UserAgent = AOL Mail
		UserAgent_F_X-Newsreader.txt	UserAgent = Mozilla Thunderbird
		UserAgent_P_Originating-Client.txt	UserAgent = Outlook
		UserAgent_Y_Message-ID.txt	UserAgent = Yahoo Mail
		UserAgent_P_User-Agent.txt	UserAgent = Outlook
		UserAgent_F_User-Agent.txt	UserAgent = Mozilla Thunderbird
		UserAgent_P_X-Mailer.txt	UserAgent = Outlook
		UserAgent_A_DomaineKey-Signature.txt	UserAgent = AOL Mail
		UserAgent_Y_DomaineKey-Signature.txt	UserAgent = Yahoo Mail
		UserAgent_F_X-Mailer.txt	UserAgent = Mozilla Thunderbird
		UserAgent_no_info.txt	null
		UserAgent_F_Originating-Client.txt	UserAgent = Mozilla Thunderbird
		UserAgent_P_X-MimeOLE.txt	UserAgent = Outlook
		UserAgent_F_X-MimeOLE.txt	UserAgent = Mozilla Thunderbird

N° cas test	N° spécification	Description	Résultat attendu
1.3.3.3	1.3.1.6, 1.3.1.8 - 1.3.1.15	Lancer les modules, lancer une analyse sur le fichier testAll.txt et privateIP.txt	Il doit y avoir avec les deux tests au moins une fois chacun des retours possibles définis dans le cahier des charges listés ci dessous et aucune clé n'a pas de valeur associée. UserAgent PrivateIP PublicIP PublicIPQualification PublicIPNetName PublicIPRange PublicIPcountryCode PublicIPcountryName PublicIPzipcode PublicIPcityName PublicIPlongitude PublicIPlatitude PublicIPmetroCode PublicIPaeraCode PublicIPdmaCode PublicIPtimeZone FirstMTAIP FirstMTAIPQualification
1.3.3.4	1.3.1.16	Lancer une analyse sur le fichier testAll.txt	Il doit y avoir les champs *PublicIPQualification1*, *PublicIPQualification2*, *FirstMTAIPQualification1* et *FirstMTAIPQualification2*. Ils doivent tous être renseignés
1.3.3.5	1.3.1.17	Lancer une analyse sur le fichier testEmpty.txt	la valeur null doit être obtenue en résultat
1.3.3.6	1.3.1.19	Lancer une analyse sur le répertoire contenant tous les fichiers de test nommé «TestAll». Ce test peut prendre un certain temps (10min)	Le temps d'exécution moyen et le temps d'exécution maximal affichés ne doivent pas dépasser les 0.86 secondes

N° cas test	N° spécification	Description	Résultat attendu
1.3.3.7	1.3.1.22-23	Renommer le fichier ConfigFindGL.xml, Lancer le programme de test, lancer les modules (ne pas oublier de le remettre par la suite). Répéter l'opération pour les fichiers ConfigFindUA.xml, whoisData.dat, Fulltrace.xml, configModuleControler.xml, GeoLiteCity.dat et partialTrace.xml. Les fichiers xml sont dans le fichier jar. Les fichiers dat sont dans le même répertoire que le fichier jar.	le lancement des modules doit échouer pour chacun des fichiers
1.3.3.8	1.3.1.8, 1.3.1.11	Lancer une analyse sur le dossier ipSource IPSource_R_MSS_IP_PR.txt IPSource_R_P_UA_Y.txt IPSource_XOI_P_UA_H.txt IPSource_PublicIP.txt IPSource_PrivateIP.txt IPSource_PublicMTA.txt IPSource_PrivateMTA.txt IPSource_MalformedIP.txt IPSource_MalformedMTAIP.txt	Les résultats suivants doivent être obtenus pour les différents fichiers PrivateIP=192.168.0.1 PublicIP=44.55.66.77 PublicIP=44.55.66.77 PublicIP=209.85.227.103 PrivateIP=192.168.1.1 FirstMTAIP=20.23.24.25 résultat null résultat null résultat null
1.3.3.9	1.3.1.22, 1.3.1.24-26, 1.3.2.3-5	Attention ce test peut prendre plusieurs heures à cause de la mise à jour des données. **ATTENTION** : Il est apparu qu'après une quantité très importantes de requêtes «whois» de mise à jour l'adresse IP publique que l'on a peut être bloquée **DEFINITIVEMENT** par certains serveurs whois. Il est possible de faire ce test sans connexion internet pour éviter ce bannissement d'adresse.	

N° cas test	N° spécification	Description	Résultat attendu
		Dans le fichier configWhoisModule.xml qui se trouve dans le jar, modifier «whoisData.dat» en «whoisDataOld.dat». Copier le fichier «whoisDataOld.dat.bak» en le nommant «whoisDataOld.dat». stopper les modules, lancer les modules et attendre la fin du lancement (retour au menu principal). Ne pas oublier de rétablir les fichiers xml à leur configuration d'origine	Il doit être indiqué dans les logs que les mises à jour pour les modules IPBL, *géo-localisation* et whois sont faites. Il est possible que la mise à jour du module whois ne soit pas terminée. Elle peut prendre plusieures heures. Les notifications de chargement réussi ou échoués apparaissent dans les logs. «*GeoIP dataSource has been not updated*» indique un échec de mise à jour de GeoLiteCity et «*GeoIP dataSource has been updated*» sa réussite. La ligne «Update whois data» est suivie de «ok» pour la réussite de la mise à jour des données whois, «failed» pour son échec et «interupted» si il a été interrompu.
1.3.3.10	1.3.2.1-2	Effacer les fichiers de logs, faire le test de la spécification 1.3.1.23	Toutes les exceptions survenues sont journalisées comme erreur
1.3.3.11	1.3.1.1 - 1.3.1.4, 1.3.1.27	Non testable sans aller contrôler le code source	
1.3.3.12	1.3.1.18	Non testable sans modification du code source	
1.3.3.13	1.3.1.20-21	Non testable	

Ceci termine l'introduction du projet. La suite du document traite de l'analyseur modulaire.

Chapitre 2

Analyseur modulaire

Dans ce chapitre est traité l'analyseur modulaire. C'est ce qui se trouve sous l'API EmailHeaderQualif , et qui englobe chacun des modules. Il y est fait une introduction sur l'analyseur modulaire puis sont données les spécifications du contrôleur de modules et de chacun des modules. Ensuite se trouve la conception de l'analyseur puis la réalisation.
L'analyseur modulaire est toute la partie cachée de l'API. C'est lui qui traite les en-têtes fournis à l'API. Il faut voir l'analyseur modulaire comme un mécanisme auquel on peut ajouter des moteurs pour en augmenter la puissance. Ces moteurs ce sont les modules. Chaque module analyse les données à disposition et en tire des conclusions. Chaque information supplémentaire fournie par un module augmente la puissance du tout. Le but premier d'un analyseur modulaire est de pouvoir ajouter facilement des modules qui augmenteront le nombre d'informations fournies en retour d'une requête tout en rendant cet ajout de puissance le moins coûteux possible au développement.
L'analyseur modulaire doit donc mettre à disposition un système de gestion de modules et un conteneur pour les données fournies pour le traitement et trouvées par les modules. De plus, il doit gérer l'appel de ces différents modules pour pouvoir répondre aux requêtes. Cependant il ne doit en aucun cas intervenir sur les données du conteneur, ceci dans le but d'externaliser un maximum le traitement aux modules. Mais encore, le gestionnaire de modules n'a aucune raison de modifier les données du conteneur puisque il ne fait pas d'analyse de données.
Le conteneur de données contient donc deux types de données : les données relatives à l'en-tête à analyser et les données trouvées.
La gestion des modules implique qu'il faut aussi gérer leurs «naissances» mais aussi leurs «morts». C'est-à-dire que ces modules peuvent avoir besoin de s'initialiser avant d'être traités ou encore de libérer les ressources qu'ils utilisent avant leur destruction. Le gestionnaire et les modules doivent donc mettre à disposition des méthodes qui permettent de faire ces initialisations et finalisations. Ceci implique que l'API elle-même doit mettre à disposition ces méthodes.
La suite donne les spécifications de l'analyseur modulaire.

2.1 Spécifications

Les spécifications données ici sont séparées en deux parties : celles du contrôleur de modules et celles des modules eux-mêmes. Ce sont les deux composantes de l'analyseur modulaire.

2.1.1 Contrôleur de modules

Ici sont listées les spécifications du contrôleur de modules. Elles sont tirées de la réflexion située en début de chapitre.

N°	Explication
2.1.1.1	Le contrôleur de module gère chacun des modules disponibles et les utilise les uns après les autres pour traiter les en-têtes
2.1.1.2	Le contrôleur gère les arbres de dépendances des modules
2.1.1.3	Le contrôleur de modules vérifie qu'il n'y a pas d'inter-dépendance dans les modules lors du chargement (pas de boucle dans le graphe des dépendances). Si il y en a, une exception doit être levée immédiatement
2.1.1.4	Le contrôleur exécute les modules selon l'arbre des dépendances (ceux qui n'ont aucune dépendances sont exécutés en premier puis, progressivement, ceux dont les dépendances ont été exécutées)
2.1.1.5	Le contrôleur ne gère pas les résultats des modules. Il ne fait que les appeler et leur passer l'objet contenant les résultats.
2.1.1.6	Le contrôleur initialise et libère les modules à la demande de *ISCIA* pour permettre à celui-ci d'initialiser les modules avant lancement (peut prendre un certain temps) et de libérer les ressources au cas où *ISCIA* devrait le faire pour quelle que raison que ce soit. La plus probable étant l'arrêt propre de ISCIA

2.1.2 Modules

Ici sont données les spécifications auxquelles chacun des modules doit répondre. Elles ont été définies selon le cahier des charges, la réflexion sur l'analyseur modulaire ci-dessus et la pré-étude.

N°	Explication
2.1.2.1	Chaque module déclare la liste des modules desquels il est dépendant
2.1.2.2	Si un champ d'information du conteneur de données utilisé par le module est manquant, on ne lève pas d'exception
2.1.2.3	Tout module devant utiliser un fichier de configuration devra le faire de façon indépendante. Il n'y a pas de fichier commun de configuration.
2.1.2.4	Tout module devra avoir une méthode d'initialisation et une méthode de libération des ressources **public void start()** pour l'initialisation et **public void stop()** pour la fermeture

2.2 Conception

La figure 1.2 (c.f. p.12) présente la structure globale du projet. Il s'y trouve l'API qui sert d'interface entre ISCIA et l'analyseur modulaire. Le nuage représente la possibilité de mettre un moyen d'utilisation à distance, par exemple *RMI*. L'analyseur modulaire se sépare en deux groupes d'éléments. Le contrôleur de module et les modules d'analyse. On peut aussi voir les différents fichiers de configuration et de données nécessaires à chacun des modules.
Voici une brève description de chacun des modules. Une description complète de chacun des modules est faite dans les chapitres suivants.

IP Verifiation C'est le module qui permet d'extraire les adresses IP d'un en-tête de courriel. Il est séparé en trois parties. Cette séparation a pour but de granulariser le codage et les tests unitaires.

[IP Analyser] Il se charge de gérer les demandes d'obtention d'IP en utilisant *IP Exctractor* et *IP Checker*

[IP Exctractor] Extrait les IP disponibles dans l'en-tête

[IP Checker] Contrôle les IP qu'on lui fournit pour éliminer les adresses incohérentes ou non pertinentes

IPBL Se charge de qualifier les IP publiques en se basant sur les listes noires utilisées pour filtrer les spams.

[IPBL Check] Essaye de qualifier les adresses IP

[IPBL Config File] Fichier de configuration du module

[IPBL Files] Fichiers contenant les listes noires

[IPBL Retriever] Se charge de mettre à jour les fichiers de listes noires

Whois Donne des informations supplémentaires sur les différentes adresses IP telles que l'intervalle dans lequel il se trouve et le propriétaire de cet intervalle

[Whois Information] Se charge d'obtenir les différentes informations à l'aide des données locales

[Whois Config File] Fichier contenant les configurations pour le module Whois

[Whois Local Informations] Fichier contenant les données utilisées par *Whois Informations*

[Whois Retriever] Se charge de mettre à jour le fichier *Whois local Informations*

User Agent Donne le *user agent* à partir d'un en-tête.

[configFindUA.xml] Fichier de configuration du module

[fullTrace.xml] Règles de détection de *user agent*

[PartialTrace.xml] Règles de détection partielles de *user agent*

IP Géo-localisation Module de géo-localisation d'une IP

[GeoliteCity.dat] Base de données contenant les informations de géo-localisation

[ConfigfileGL.xml] Fichier de configuration du module

En plus des éléments cités ci-dessus, le schéma 1.2 montre le gestionnaire de modules «Module Controler» et son fichier de configuration qui contient la liste des modules.

2.2.1 Principes de fonctionnement de l'analyseur modulaire

Voici la liste des différents principes de fonctionnement de l'analyseur modulaire :

1. Initialisation et arrêt propre des modules
2. Gestion des dépendances entre les modules
3. Gestion des priorités d'exécution des modules
4. Ajout dynamique de modules

2.2.1.1 Initialisation et arrêt propre des modules

Certains modules devant disposer de ressources (par exemple chargement de données en mémoire à partir d'un fichier), ces ressources doivent souvent être initialisées. C'est pourquoi il faut un moyen d'initialiser les différentes ressources. Le gestionnaire de modules mets à disposition une méthode qui permet de lancer cette initialisation. Si cette initialisation n'a pas été lancée, toutes requêtes pour obtenir des informations lèvera une exception.
De même que les ressources doivent être initialisées, elles doivent souvent être libérées à l'arrêt du programme. C'est pourquoi l'analyseur fournit aussi une méthode de libération des ressources. Après l'exécution de cette méthode, toute requête d'obtention d'information sera refusée à moins que la méthode d'initialisation ne soit appelée.
Pour garantir l'implémentation de ces méthodes, tous les modules ont comme point d'entrée une classe héritant d'une classe abstraite «*AbstractModule*» qui définira ces deux méthodes.

2.2.1.2 Gestion des dépendances entre les modules

Les modules peuvent posséder des dépendances sur d'autres modules, c'est-à-dire qu'ils peuvent avoir besoin des données fournies par d'autres modules. Ceci implique que les modules dont ils dépendent doivent être exécutés avant eux. Cependant, si il y a des dépendances circulaires entre les modules, cela posera problème durant le traitement des requêtes sur les en-têtes car il sera impossible de déterminer lequel des modules exécuter en premier dans la dépendance circulaire. Pour éviter cela, le gestionnaire de modules contrôle qu'il n'y ait pas de dépendances circulaires lors de l'initialisation des modules. Si il y a une dépendance circulaire, l'initialisation échoue et une exception est levée. Le détail de cette gestion est expliqué dans la partie réalisation de ce chapitre.

2.2.1.3 Gestion des priorités d'exécution des modules

Les modules n'ont pas de priorités à proprement parlé. Cependant, les dépendances implique que certains modules doivent être exécutés avant d'autres. C'est ce qu'on entend ici par priorité.
Pour chaque requête, tous les modules doivent être appelés dans un ordre qui fait que les modules sans dépendances sont exécutés en premier et petit à petit les modules dont les dépendances ont déjà été exécutées sont exécutés à leur tour.

2.2.1.4 Ajout dynamique de modules

ISCIA étant redémarré très peu souvent et, in extenso EmailHeaderQualif aussi, il est nécessaire de pouvoir insérer un nouveau module sans pour autant devoir re-compiler tout EmailHeaderQualif . Pour ce faire, un fichier de configuration contenant la liste des modules est utilisé par le gestionnaire de modules. Quand le gestionnaire de modules démarre, il va vérifier ce fichier pour charger les modules. Il suffit donc de redémarrer le gestionnaire de module pour prendre en compte le nouveau module.

2.2.2 Principes de fonctionnement des modules

En plus de leur fonction d'analyse, les modules répondes à ces différents principes :

1. Toutes les données nécessaires au traitement sont chargées en mémoire

2. La mise à jour des données en mémoire doit être gérée par les modules

2.2.2.1 Données de traitement en mémoire

Pour des raisons d'efficacité (c.f. Spécification n° 1.3.1.19), toutes les données nécessaires aux modules doivent être chargées en mémoire. Un tel fonctionnement peut paraître dangereux d'un point de vue de la place mémoire utilisée. Cependant, si un accès aux fichiers ou à un service externe est fait à chaque nouvelle requête, le temps d'exécution sera décuplé par les temps d'accès, ce qui n'est pas tolérable au vu du nombre de requêtes à traiter par jour.

2.2.2.2 Gestion des données en mémoire

Comme dit au début du chapitre, le gestionnaire de modules doit gérer les différents modules. Cependant il ne doit pas avoir à agir pour ou sur les données nécessaires au fonctionnement des différents modules. Les deux seuls moments où son intervention a des implications sur ces données, c'est lors de l'initialisation et de la finalisation. Cependant, il se peut que l'initialisation ne soit faite que très rarement puisque ISCIA ne redémarre pas souvent. Mais les sources de données initiales (par exemple la base de donnée des informations de géo-localisation) peuvent changer plus souvent que ISCIA ne redémarre. Il faut donc que les modules gèrent la mise à jour régulière de leurs données en mémoire. Pour ce faire, ils doivent d'eux-mêmes, et sans bloquer les requêtes, mettre à jour leurs données en mémoire.
Si la mise à jour ne doit pas bloquer les requêtes c'est parce qu'il ne faut pas augmenter le temps de traitement des requêtes ou bloquer *ISCIA* sur une mise à jour.

2.2.3 Structure de l'analyseur modulaire

La structure de l'analyseur modulaire découle des raisonnements précédents. Ci-après se trouve un diagramme de classe simplifié de l'analyseur modulaire. S'ensuit une explication détaillée de chacune des classes présentées dans se schéma.

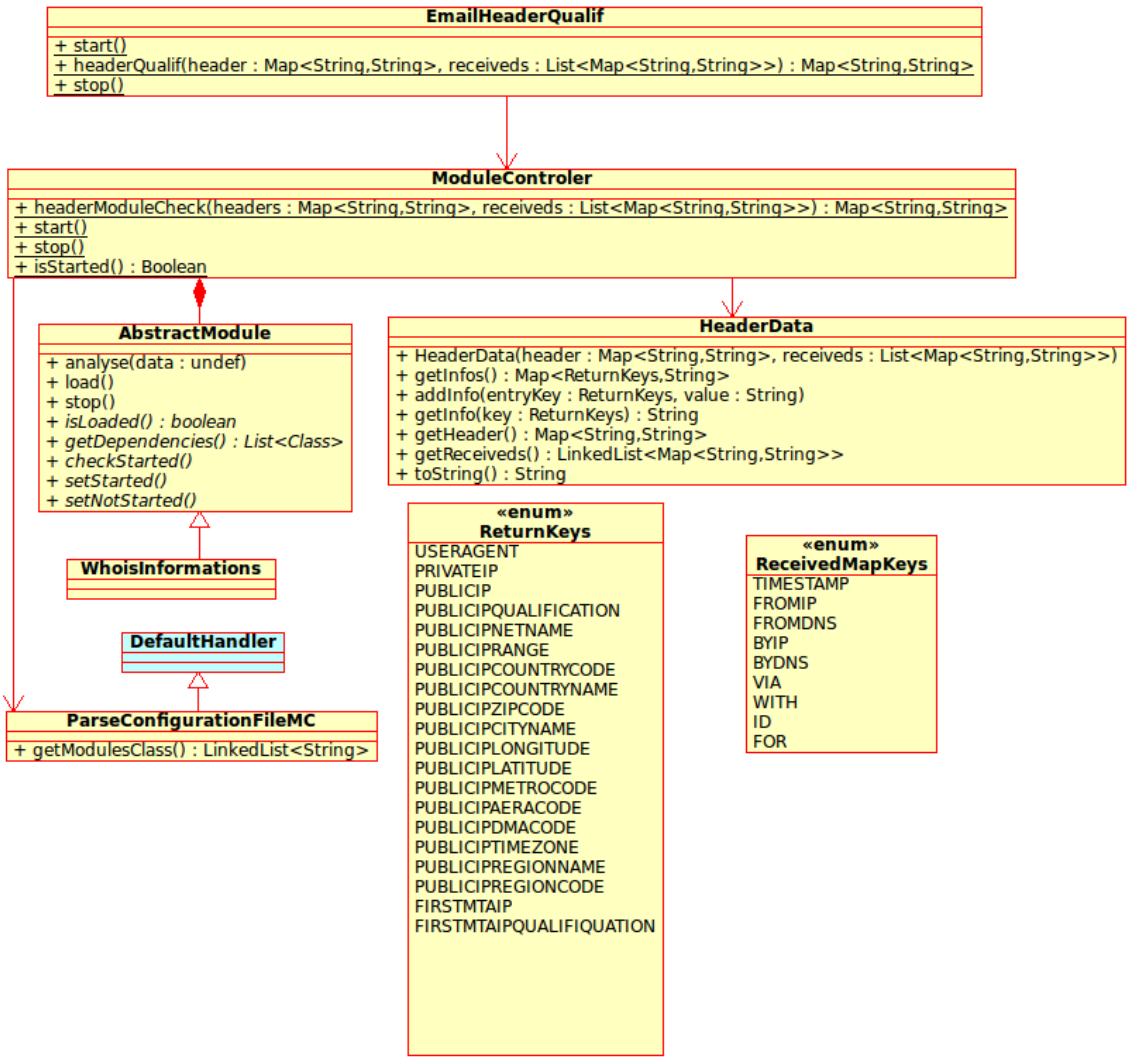


FIGURE 2.1 – Schéma UML simplifié de l'analyseur modulaire

Il est aisé de faire le parallèle avec le schéma 1.2 page 12. La classe «`EmailHeaderQualif` » représente l'API et la classe «`ModuleControler`» le contrôleur de modules. La classe «`WhoisInformations`» représente le point d'entrée de «`ModuleControler`» pour le module Whois. Tout module doit posséder une classe du même type. C'est-a-dire une classe qui étend la classe «`AbstractModule`».

2.2.3.1 Classe EmailHeaderQualif

La partie visible de l'extérieure, ou publique est la classe «`EmailHeaderQualif`». Cette classe n'est pas instanciable et toutes ses méthodes sont statiques car elle ne sert que d'interface à l'extérieur et n'a aucune raison d'être instanciée. On réduit ainsi la quantité de mémoire nécessaire et accélère l'accès aux méthodes.
«`EmailHeaderQualif`» défini trois méthodes :

start Son but est d'initialiser les différents modules qui pourraient demander un chargement préalable avant d'être exécutés.

stop À l'inverse de start, elle permet de libérer correctement chacune des ressources utilisées par les différents modules.

headerQualif C'est la méthode qui permet d'obtenir les différents informations via les modules disponibles. Elle prend en paramètre le header et la liste des *received*s contenus dans l'en-tête.

Cette classe ne fait qu'interfacer avec la classe «`ModuleControler`».

2.2.3.2 Classe ModuleControler

Cette classe est le centre névralgique de l'analyseur modulaire. C'est elle qui gère les modules et les requêtes reçues. Son fonctionnement est en soi assez simple. Les méthodes publiques fournies par cette classe sont les mêmes que pour 2.2.3.1. Tout le code de cette classe est défini en statique afin d'optimiser la vitesse d'exécution et le constructeur de la classe n'est pas accessible pour qu'elle ne puisse pas être instanciée. En effet, il ne servirait à rien d'instancier cette classe car tous les modules sont aussi défini en statique (voir 2.3.1). De plus, il ne peut y avoir d'appel concurrent sur cette classe comme défini dans la spécification 1.3.1.20.

2.2.3.2.1 Gestion de l'état des modules L'un des rôles de cette classe est de gérer l'état des différents modules. Ceux-ci peuvent avoir deux états. C'est également le cas du gestionnaire de module. Il doit tout d'abord être initialisé, ce qui se fait avec la méthode **start()**. Si cette méthode n'est pas appelée en premier lieu, toute requête aboutira sur une exception.
Le module possède au final deux états : démarré et arrêté. La suite de cette section liste les différents états et événements (appels de méthodes) et leurs conséquences.

État démarré

start N'a aucun effet

stop Arrête chacun des modules en appelant sa méthode **stop()** (c.f. 2.2.3.3.3) et passe dans l'état arrêté

headerModuleCheck Lance l'analyse de la requête sur chacun des modules (c.f. 2.2.3.3.1)

État arrêté

start La séquence d'initialisation est lancée (c.f. 2.2.3.2.2) puis le gestionnaire passe en état démarré

stop Aucun effet

headerModuleCheck Lève une exception. Le gestionnaire doit être lancé pour appeler cette méthode

2.2.3.2.2 Séquence d'initialisation La séquence d'initialisation comprend plusieurs phases :

1. Lecture du fichier de configuration
2. Construction du graphe des dépendances des modules
3. Vérification de l'absence de cycles dans le graphe des dépendances
4. Définition des priorités d'exécution des modules selon les dépendances
5. Démarrage des modules selon l'ordre d'exécution préalable

Les points 2 et 3 sont explicités dans la réalisation.
À chacune de ces étapes peuvent survenir des erreurs. Si cela arrive, le démarrage est annulé, les modules précédemment démarrés sont arrêtés, toutes les erreurs sont journalisées.

2.2.3.2.3 Traitement des requêtes Le diagramme d'activité ci-après représente le traitement effectué par le gestionnaire de module pour les requêtes reçues.

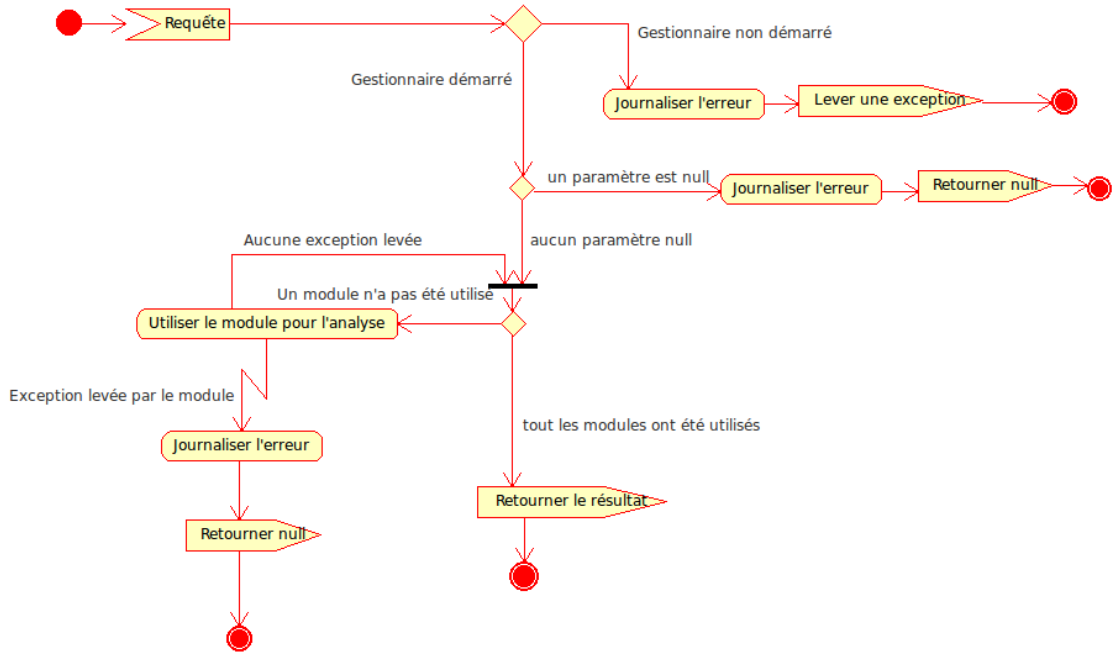


FIGURE 2.2 – Diagramme d'activité de l'analyse d'une requête par le gestionnaire de modules

Il est montré ici que le seul moment où cette partie de l'analyseur modulaire lève une exception c'est quand il n'a pas été préalablement démarré. Les exceptions levées par les différents modules ne sont pas prises en compte, ni la mauvaise formulation de requête (paramètre null).

2.2.3.3 Classe AbstractModule

Ce paragraphe traite de la classe abstraite que doit implémenter tout module. Le but de cette classe est d'avoir un moyen uniforme pour le contrôleur de modules de gérer les modules. Les paragraphes suivants donnent les méthodes et leurs buts. Il est important de comprendre que le fonctionnement des modules est soumis aux mêmes états que le contrôleur de modules (c.f. 2.2.3.2).

2.2.3.3.1 void analyse () Cette méthode est celle appelée pour faire une requête sur un header. Elle prend les mêmes paramètres que la fonction headerQualif mais prend en plus le conteneur de données trouvées (c.f. 2.2.3.4).

2.2.3.3.2 void load () Cette méthode permet de charger le module et le met en état démarré. Toute erreur de chargement du module lève une exception.

2.2.3.3.3 void stop () Cette méthode permet d'arrêter le module.

2.2.3.3.4 Boolean isLoaded () Cette méthode renvoie un booléen qui est à vrai si le module a été chargé.

2.2.3.3.5 List<Class> getDependencies () Cette méthode donne la liste des modules desquels est dépendant le module.

2.2.3.4 Classe HeaderData

Cette classe contient les informations trouvées par les différents modules. Elle permet au gestionnaire de module de faire complétement abstraction du traitement fait par les modules. «`HeaderData`» contient une table de hachage avec les différentes informations trouvées. Les clés qui sont dans cette tables doivent correspondre aux retours définis dans le cahier des charges.
La seule interaction qu'a le contrôleur de modules avec la classe «`HeaderData`», c'est quand il doit retourner les résultats de l'analyse suite à une requête. À ce moment uniquement il utilise la classe «`HeaderData`» pour obtenir la table de hachage des résultats
La prochaine section traite de la façon dont a été codé l'analyseur modulaire.

2.2.3.5 Classe ParseConfigurationFileMC

Cette classe défini permet de parser le fichier XML de configuration. Elle étend la classe «`DefaultHandler`» pour être passée à un parseur XML présent par défaut dans la machine virtuelle Java™. Les informations sur la structure du fichier de configuration se trouvent en 2.2.5.

2.2.3.6 Enumeration ReturnKeys

Cet *enum* donne la liste des clés que retourne EmailHeaderQualif . Il permet de centraliser ces informations et de ne pas faire d'erreurs de codage sur des chaînes de caractères et de modifier facilement les retours possibles.

2.2.3.7 Enumeration ReceivedMapKeys

Cet *enum* donne la liste des clés disponibles dans les *received* passés en paramètre de la méthode headerQualif . Il permet de centraliser ces informations et de ne pas faire d'erreurs de codage sur des chaînes de caractères et de modifier facilement les retours possibles.

2.2.4 Traitement des dépendances

La gestion des dépendances tel que décrite en 2.2.3.2.2 implique l'utilisation d'un graphe orienté de dépendances. Chaque module est un sommet et chaque dépendance est un arc orienté.

Pour ce faire, une matrice représentant le graphe a été utilisée. Pour pouvoir résoudre[1] les dépendances il a fallu développer un algorithme qui élimine progressivement les sommets dont aucun autre module ne dépend (représenté par l'élimination de ses arcs). Si à la fin de l'algorithme il reste des arcs dans le graphe (indiquant ainsi la présence d'au moins deux sommets) c'est qu'il y a un cycle.

L'exemple ci-dessous représente le traitement de trois modules A, B et C. Le module C a une dépendance sur le module B et le module B a une dépendance sur le module A.
Matrice du graphe :

$$M1 = \begin{array}{c} \\ A \\ B \\ C \end{array} \begin{array}{c} \begin{array}{ccc} A & B & C \end{array} \\ \begin{pmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix} \end{array}$$

Quand une colonne ne contient que des O cela signifie que le module correspondant à cette colonne n'a aucune autre module qui dépends de lui (le module C dans l'exemple ci-dessus). Donc si on enlève les dépendances de ce module on ne risque pas de casser un cycle existant. Ce qui donne le graphe suivant pour l'exemple ci-dessus.

$$M2 = \begin{array}{c} \\ A \\ B \\ C \end{array} \begin{array}{c} \begin{array}{ccc} A & B & C \end{array} \\ \begin{pmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} \end{array}$$

En répétant l'opération pour B on arrive à la matrice suivante :

$$M3 = \begin{array}{c} \\ A \\ B \\ C \end{array} \begin{array}{c} \begin{array}{ccc} A & B & C \end{array} \\ \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} \end{array}$$

Il n'y a plus d'arc dans le graphe. Ce qui implique qu'il n'y a pas de dépendances circulaires. En prenant un cas de dépendance circulaire où A dépends de B et inversement on obtient la matrice suivante :

1. On entend par résoudre : vérifier l'absence de cycle dans le graphe

$$M4 = \begin{array}{c} \\ A \\ B \\ C \end{array} \begin{array}{c} \begin{array}{ccc} A & B & C \end{array} \\ \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} \end{array}$$

Enlever les dépendances à C ne change pas la matrice. Il est impossible d'enlever les dépendances de B ou de A. On arrive donc à un graphe qui comporte toujours des arcs ce qui indique une dépendance circulaire. C'est comme ceci que fonctionne l'algorithme de résolution des dépendances.

De plus, durant cette résolution, il faut trouver l'ordre d'exécution des graphes pour éviter les erreurs de dépendances durant le traitement des requêtes. Pour ce faire, il suffit qu'à chaque fois que les dépendances d'un module sont enlevées, il soit considéré comme un module a exécuter avant les modules ayant encore une ou plusieurs dépendance. On obtient donc ainsi un ordre d'exécution.

Tout le développement et le test de cet algorithme a un coût. Cependant, en considérant que l'ajout de module dans le projet n'est pas vraiment un événement qui survient souvent (au mieux à quelques mois d'intervalle), le temps pris à développer cette fonctionnalité est un peu exagéré. Le contrôle des dépendances à la main avant chaque ajout de nouveau module aurait été moins coûteux. Cependant cette considération n'a été faite qu'après le développement de cet algorithme. Il est donc en place.

2.2.5 Fichier de configuration

Le fichier de configuration du module est au format XML(voir [29] pour plus d'informations sur ce langage de description) car toutes les fichiers de configuration dans *ISCIA* le sont. Ceci permet de garder une uniformité vis-à-vis de *ISCIA*. Cette uniformité est gardée pour tout les fichiers de configuration du projet.
Voici un exemple de structure de ce fichier :

```
<configurationModuleControler>
        <moduleClassList>
                <module>ch.iict.iscia.ehq.modules.whois.
                    WhoisInformations</module>
                <module>ch.iict.iscia.ehq.modules.geoloc.
                    GeoLocModule</module>
        </moduleClassList>
</configurationModuleControler>
```

En premier il y a l'identifiant «configurationModuleControler» qui est la base de la structure du fichier. Ensuite il se trouve «moduleClassList» qui précise la structure de liste des modules que doit utiliser l'analyseur modulaire. Finalement il y a une liste de «module». Dans un de ces champs se trouve le chemin complet de la classe étendant «`AbstractModule`» qui peut être utilisée comme point d'entrée d'un module. Attention, il est entendu par chemin complet le nom du paquetage suivi du nom de la classe.

2.3 Réalisation

La conception ne tient pas toujours compte des parties inhérentes au codage. Cette section traite des différents points qui n'entre pas dans la conception mais qui interviennent dans le développement de l'analyseur modulaire.
Ici se trouvent les informations sur l'optimisation du code et sur le moyen d'invoquer les méthodes des différents modules à partir du nom de leur classe. Mais encore il est discuté de l'implémentation des différents énumérés utilisés dans l'analyseur modulaire.

2.3.1 Optimisation du code

Presque toutes les méthodes définies par les classes sont statiques. La raison d'un tel choix est d'optimiser la vitesse de traitement, car une méthode statique est plus rapide à appeler qu'une méthode sur un objet. De plus, seules les classes de modules sont instanciables, toujours pour les mêmes raisons d'optimisation de vitesse mais aussi de mémoire. En effet une classe non instanciable ne peut générer d'objets inutiles. Et vu qu'il n'y a pas d'appels concurrents sur l'API (c.f. spécification 1.3.1.20), il n'y a aucune raison d'instancier des objets.
Les seules classes instanciables sont les modules. Cependant, afin de limiter les risques d'instanciation surnuméraires, derrière les instances de modules sont utilisées des méthodes statiques pour les mêmes raison d'optimisation que celles citées précédemment.

2.3.2 Méthode de traitement de requêtes sur les modules

Pour pouvoir accéder aux modules «Module Controler» ne dispose que de la Classe des modules et du nom de la méthode à invoquer. Ceci implique qu'il faut faire de la réflexion java.
La réflexion java permet de traiter les classes et les méthodes comme des objets. C'est à dire d'avoir un objet qui représente une classe et d'instancier cette classe à partir de cet objet. Une fois l'objet instancié il est possible d'invoquer une méthode de cet objet en utilisant simplement le nom textuel de cette méthode. Par exemple en utilisant la chaîne de caractères "toString" pour invoquer la méthode **toString()** de l'instance d'un objet.
Les risques inhérents à ce type d'invocation sont que la classe n'est pas réellement celle qu'il faudrait (dans notre cas qui n'étend pas la classe «AbstractModule» ou que la méthode invoquée n'existe pas. Auquel cas des exceptions sont levées. Il faut donc gérer ces risques d'erreurs dans le code.
La gestion des erreurs dues à la réflexion ne doit pas remonter jusqu'à l'appelant de Email-HeaderQualif . En effet une requête ne doit jamais renvoyer une erreur sauf si l'analyseur modulaire n'a pas démarré.
La liste des classes dans le fichier de configuration peut être fausse. Ce qui a pour conséquence de lever une exception lors de l'analyse avec la réflexion. C'est pourquoi il est contrôlé lors du démarrage que les classes définies dans le fichier de configuration sont existantes et répondent aux critères des modules.

2.3.3 Journalisation

La journalisation se fait avec log4j. Un fichier de configuration «log4j.properties» contient toute la configuration de journalisation de EmailHeaderQualif .

La façon dont sont géré les entrées de journalisation est importante de deux points de vue. Le premier étant le risque de surcharge des disques durs avec des entrées de journal qui ne sont pas pertinentes. Le second est l'affichage utilisateur durant les tests.
La politique de journalisation utilisée prenant en compte les deux points précédents est la suivante :

- Les entrées de journal de type "debug" ne sont pas enregistrées. Elles ne sont là que pour aider au développement et ne doivent pas surcharger les fichiers de log.
- Toutes les entrées de journal de type supérieur à "debug" sont enregistrées dans un fichier de log. Son nom est défini dans le fichier de propriétés. Ceci pour permettre de corriger des problèmes pouvant survenir.
- Pour les besoins des tests et de la validation toutes les entrées supérieures à debug sont aussi affichées dans la console. Ceci peut facilement être modifié par la suite dans le fichier de configuration de log4j pour que ces entrées n'apparaissent plus.
- Seules les erreurs empêchant un module ou le gestionnaire de module de démarrer sont journalisées. Les autres erreurs ne le sont pas.

2.3.4 Implémentation des énumérés de référence

Les énumérés permettent de référencer les différentes clés présentent dans les «Map» de *received*s passés en paramètres et les clés données dans le «Map» de retour de EmailHeaderQualif .
Pour permettre d'y associer une chaîne de caractères il a fallu créer un constructeur pour la classe représentant l'enum qui prend en paramètre un «String» et avoir un champ «String» dans la classe permettant d'associer une entrée du type énuméré avec une chaîne de caractères.
Finalement la méthode **toString()** a été implémentée pour permettre d'obtenir cette chaine de caractères.
On obtient ainsi une implémentation du type énuméré qui associe chaque valeur de l'énuméré à une chaîne de caractères. Ceci permet d'éviter d'utiliser des chaînes de caractères directement dans le code et limite le risque d'erreurs au codage. Mais surtout cela permet de pouvoir modifier ces valeurs de façon centralisée sans avoir l'obligation de vérifier tout le reste du code.

2.4 Tests unitaires

Les tests unitaires ont été réalisés de façon à contrôler qu'aucune erreur de logique n'apparaisse dans le code. Ci-après sont listés les différentes choses testées dans les tests unitaires selon les différentes classes.

La classe «EmailHeaderQualif » n'est pas testée dans les tests unitaires. Elle ne fait qu'appeler le gestionnaire de modules et son fonctionnement est contrôlé durant les tests manuels. De plus, la vérification de son bon fonctionnement par des tests unitaires coûterait plus cher en temps de développement qu'en temps de test. Cela reviendrait à faire les tests de validation directement dans les tests unitaires.

La classe «ParseConfigurationFileMC» est contrôlée durant les tests manuels. Il n'y a pas de tests unitaires la concernant.

2.4.1 Tests unitaires de «HeaderData»

Premièrement, les contrôles sur le constructeur s'assurent de lever une exception si l'en-tête ou les les receiveds passés en paramètres sont *null* et qu'ils ne sont pas modifiés.

Deuxièmement, on s'assure que l'ajout d'informations (méthode **addinfo(returnkey ReturnKeys)**) fonctionne et qu'il ne modifie pas l'en-tête et les received contenus dans HeaderData. Puis il est contrôlé que l'insertion d'une information avec une clé ou une valeur nulle lève une exception.

Finalement il est contrôlé qu'une information insérée est retournée telle qu'elle a été entrée.

2.4.2 Tests unitaires de ModuleControler

Les tests unitaires sont faits avec [38]. Junit permet de mettre en place un système de tests pour chaque classe. Une classe de test corresponds à une classe à tester. Il est alors possible de tester chacune des méthodes publiques d'une classe. Maven permet de lancer les tests unitaires avant d'exécuter la phase "install". Cette phase est la création finale des fichiers jars et des ressources nécessaires. Ainsi on garantit, au minimum, que les tests unitaires sont validés avant de créer la «cible» finale du projet.
Premièrement les tests unitaires du gestionnaire se font sans modules activés dans le fichier de configuration pour ne tester que le contrôleur et son fonctionnement. Ceci afin de granulariser les tests. Les tests manuels et les tests de validation contrôlent que l'interaction avec les modules est correcte.

Il est vérifié que l'état du contrôleur corresponde à ce qui est défini en 2.2.3.2 et que la méthode **headerModuleCheck** lève bien une exception si le module n'est pas démarré et qu'elle n'en lève pas dans le cas contraire.

2.4.3 Test manuel

Les tests manuels sont les tests effectués à la main qui ne peuvent être faits sans modifier de code. Un seul test doit être effectué à la main. C'est celui des dépendances circulaires. Il a été fait comme suit :

Ajouter une dépendance circulaire entre le module IPVerification et le module UserAgent puis essayer de démarrer EHQ. À la suite du démarrage trois messages apparaissent dans la console : «There are cycled dependencies in the modules», «Exception occured while starting modules. Stopping!» et finalement «Stopping modules : OK». Puis l'exception «RejectedExecutionException» levée doit apparaître avec le message «Error while starting the modules. Check modules dependencies».

Ce test a fonctionné.

Tous les autres tests sont des tests de validation car il prennent en compte l'intégralité du fonctionnement de EmailHeaderQualif . Ils ne concernent plus uniquement l'analyseur modulaire ou sont pris dans les cas-tests.

Le chapitre suivant traite du module IP Verification.

Chapitre 3

Module IP Verification

Ce chapitre traite du module d'extraction et de vérification des adresses IP. Une première analyse a été faite puis une conception des différentes parties du module a été effectuée. La partie réalisation de ce chapitre expose les différents problèmes rencontrés durant son développement.

3.1 Analyse

Le but de ce module est de remplir les spécifications 1.3.1.8, 9 et 11, donc de pouvoir extraire l'adresse IP publique ou privée de la machine émettrice d'un courriel et du premier MTA de ce courriel. Pour ce faire le travail [14] permettant d'extraire des adresses IP source a été utilisé comme base de travail. Il a dû être adapté car la détection d'adresses IP est limitées à trois règles qui sont très restrictives. Le chapitre 3 de [14] a été utilisé comme base de travail pour l'analyse ainsi que la pré-étude faite dans ce travail.

Les points importants de [14] et de la pré-étude sont les suivants :
- Les trois premières règles de détection définie dans [14] p.24 citées ci-après
- Utilisation des deux premiers *received* insérés dans l'en-tête par les MTA. Pour essayer d'obtenir l'adresse IP publique ou privée de la machine émettrice et du premier MTA.
- Contrôle du type d'adresse IP (publique, privée ou spéciale)

Règles de détection définies dans [14] :

1. Les mails envoyés par l'agent utilisateur « Hotmail » contenant l'en-tête « X-Orignating-IP » dont l'adresse IP est publique et que le champ « With » de l'en-tête « received » plus ancien contient la valeur « http », l'adresse IP source est alors l'adresse contenu dans le champ « X-Orignating-IP».
2. Les mails envoyés par l'agent utilisateur « Yahoo Mail » contenant dans le champ « IPBy » de l'en-tête « recieved » le plus ancien une adresse IP publique et que le champ « Via » de ce même en-tête « received » contient la valeur « http », l'adresse IP source est alors l'adresse contenu dans le champ « IPBy ».
3. Les pourriel contenant la mention « MICROSOFT SMTP SERVER » (non sensible à la casse) dans le champ « with » de l'en-tête « received » le plus ancien, contient

l'adresse IP source privée dans le champ « IPBy » de ce même en-tête « received ».

La première adaptation à apporter à [14] est qu'il ne permet d'extraire que les adresses IP de la source d'un courriel. Il faut donc rajouter un traitement pour pouvoir extraire l'adresse du premier MTA et pouvoir extraire plus d'adresses IP que celles des trois règles citées précédemment.

3.1.1 Analyse de l'extraction d'adresse de la machine Émettrice

Hormis les trois règles citées précédemment, il est possible d'obtenir des adresses IP dans d'autres cas.

Fondamentalement, le but de l'extraction d'adresses IP à partir d'en-têtes permettra par la suite de recouper des adresses de plusieurs pourriels pour obtenir des informations sur la source d'une série de courriels frauduleux. Il faut donc essayer d'extraire un maximum d'adresses pour pouvoir permettre de recouper ces données. Il est clair qu'il est possible que certaines parties des en-têtes soient forgés. Cependant il est difficile de déterminer à quel niveau cela ce situe.

Finalement le traitement d'extraction d'adresse est simple. Dans l'en-tête suivant on trouve «20.23.24.25» et «209.85.227.103» deux adresses e-mail qui peuvent être potentiellement celle du premier MTA(celle du haut) et celle de la machine ayant émis le courriel (celle du bas). Ce sont toutes les deux des adresses qui apparaissent dans le champ *«from»* du premier et du second received (dans leur ordre d'insertion par les MTA.

```
Received: from fep18.mx.upcmail.net ([62.179.121.38]) by bay0-mc2
   -f20.Bay0.hotmail.com with Microsoft SMTPSVC(6.0.3790.4675);
         Thu, 18 Nov 2010 09:42:59 -0800
Received: from edge04.upcmail.net ([20.23.24.25])
          by viefep18-int.chello.at
          (InterMail vM.8.01.02.02 201-2260-120-106-20100312)
              with ESMTP
          id <20101118174255.NJVN1353.viefep18-int.chello.
              at@edge04.upcmail.net>;
          Thu, 18 Nov 2010 18:42:55 +0100
Received: from dftr5yre ([209.85.227.103])
        by edge04.upcmail.net with edge
        id YhiZ1f02r4XpT8904hibsX; Thu, 18 Nov 2010 18:42:55
           +0100
```

On obtient donc les résultats en analysant les adresses IP contenues dans le champ *«from»* des deux premiers received.

Le deuxième risque est que les adresses trouvées soient non valides (des adresses ip impossibles telles que « 500.78.34.12 ») ou du mauvais type. Par exemple une adresse IP privée pour le premier MTA. Dans ce cas ces adresses ne doivent pas être retournées.

Une adresse IP de machine source doit donc posséder les caractéristiques suivantes pour être prise en compte par ce module :

– Être valide
– Être publique ou privée

L'adresse du premier MTA doit quand à elle possé les caractérisiques suivantes :

– Être publique
– Être valide

3.2 Conception

On peut extraire de l'analyse deux parties à ce module. Premièrement la partie d'extraction des adresses IP et deuxièmement la partie de contrôle des adresses IP. Cette séparation du traitement apporte deux avantages.

Le premier est la possibilité de séparer le développement et les tests unitaires en deux parties distinctes. Le deuxième et le plus important, est de pouvoir utiliser le contrôle d'adresses IP dans d'autres modules par la suite.

De ces constatations et des prérequis définit pour les modules découle le diagramme de classe suivant :

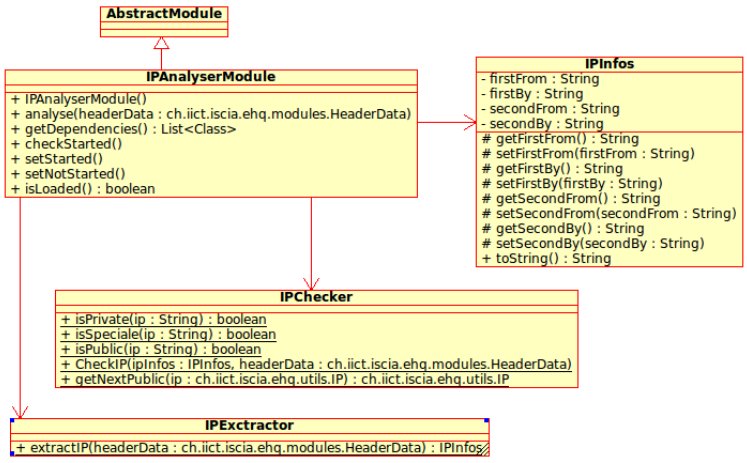


FIGURE 3.1 – Schéma UML de IP Verification

La suite liste les différentes classes et ce qu'elle font.

3.2.1 Classe IPAnalyserModule

C'est la classe d'entrée du module qui est utilisée par le gestionnaire de modules. Elle étends donc la classe «`AbstractModule`». Elle ne possède rien de particulier car, même si elle en implémente les méthodes, elle ne nécessite pas de démarrage ou d'arrêt (méthodes **load** et **stop** vu qu'elle ne possède aucune ressource à charger ou à libérer.

La méthode **analyse** telle que définie dans «`AbstractModule`» va extraire les informations en utilisant «`IPExctractor`» et va les valider en utilisant «`IPChecker`».

3.2.2 Classe IPInfos

Conteneur pour les adresses IP trouvées dans les deux premiers received, cette classe est utilisée dans la méthode **analyse** de «`AbstractModule`». Elle permet de faire le transfert des adresses IP trouvées dans «`IPExtractor`» à «`IPChecker`». Y sont stockées les différentes adresses IP suivantes :

- Adresse IP contenue dans le from du premier *received*
- Adresse IP contenue dans le by du premier *received*
- Adresse IP contenue dans le from du deuxième*received*
- Adresse IP contenue dans le by du deuxième *received*

Aucune vérification n'est faite sur ces adresses IP dans cette classe. Elles sont juste stockée dans le conteneur sous forme de «`String`».

3.2.3 Classe IPExctractor

Fondamentalement cette classe est à voir comme une librairie. En effet, la méthode définie ici est statique et cette classe n'est pas instanciable. Ce choix a été fait pour des raisons de performances mais aussi car il est inutile d'instancier cette classe qui ne contient aucun attributs.
Elle ne possède qu'une seule méthode : **IPInfos extractIP (headerData HeaderData)**. Cette méthode extrait simplement les adresses IP contenues dans les champs «from» et «by» des deux premiers received sans autre forme de contrôle et les renvoie dans un objet «`IPInfos`». Cette méthode inclut aussi les règles de détection du travail de R. Cornu [14] citées précédemment pour obtenir un maximum d'adresses IP.

3.2.4 Classe IPChecker

Pour les mêmes raisons que «`IPExtractor`» toutes les méthodes de cette classe sont définies en statique. Le code de cette classe a été repris de celui de [14]. Son but est de pouvoir vérifier des adresses IP. Ses méthodes sont les suivantes :
boolean isPrivate (String ip) retourne vrai si l'adresse IP fournie en paramètre est valide et privée.

boolean isSpeciale (String ip) retourne vrai si l'adresse IP fournie est valide et dans la catégorie «spéciale» des intervalles d'adresses IP.

boolean isPublic (String ip) retourne vrai si l'adresse IP n'est ni spéciale ni privée.

void checkIP (IPInfos infos, HeaderData headerData) permet de contrôler les adresses IP extraites et d'ajouter les adresses IP trouvées et validées dans headerData.

IP getNextPublic (IP ip) a été rajoutée pour les besoins du module whois. Elle permet d'obtenir la prochaine adresse IP publique à partir d'une adresse IP en se basant sur les intervalles d'adresse qui sont définit dans la classe. Retourne null si il n'y en a pas.

Pour pouvoir contrôler les différentes adresses IP, la classe se base sur deux listes d'intervalles définis en statique dans la classe : une liste d'intervalles privés et une liste d'intervalles spéciaux.
Le fichier fournit par IANA sur `http://www.iana.org/assignments/ipv4-address-space/ipv4-address-space.xml` donne la liste des intervalles d'adresse et leur attribution (publique, privée ou spéciale). Il a été utilisé comme base pour corriger les intervalles privées ou spéciaux définis dans IPChecker

3.3 Réalisations

La seule difficulté rencontrée durant la réalisation de ce module a été de faire les tests unitaires pour les intervalles d'adresse IP. La première méthode de test consistait à tester l'ensemble de chacun des intervalles pour les trois méthodes **isPrivate**, **isPublic** et **isSpeciale**. Evidemment le temps de développement et surtout celui de test est beaucoup trop élevé. Finalement il n'est testé que le début, la fin et une adresse IP au milieu pour chacun des intervalles privés et spéciaux ainsi qu'une petite série d'adresse IP publique.

3.4 Tests

Les tests unitaires effectués pour ce module sont définis comme suit :

«`IPAnalyserModule`» Cette classe n'a pas de tests unitaires. Elle ne fait que faire le pont entre le gestionnaire de module et «`IPChecker`» et «`IPExctractor`» et elle n'a pas besoin d'être démarrée.

«`IPExctractor`» Il est simplement vérifié qu'avec un «`HeaderData`» donné les bonnes adresses IP sont retournées dans le «`IPInfos`» de retour pour **extractIP**.

«`IPChecker`» Les méthodes **isPrivate**, **isPublic** et **isSpeciale** sont testées pour chacun des intervalles définis comme privés ou spéciaux. Pour chaque intervalle on test sa première adresse, sa dernière et une autre adresse située dans cet intervalle. Des tests sont aussi effectués pour chacune des méthodes avec des adresses IP qui ne sont pas valides et

des adresses IP publiques.

Le chapitre suivant traite du module de géo-localisation d'adresses IP.

Chapitre 4

Module de géo-localisation

La géo-localisation d'adresse IP est traitée dans ce chapitre pour pouvoir localiser les adresses IP de machine émettrice de courriel.
L'intégration du travail de R. Cornu [14] implique d'adapter la géo-localisation d'adresse IP qui y a été faite. Il ne sera pas traité de la façon dont est faite cette géo-localisation car c'est un travail qui a déjà été fait au préalable. Cependant il convient de lister les prérequis et adaptations à apporter à la géo-localisation pour qu'elle s'intègre correctement au système modulaire mis en place. Ceci est fait dans l'analyse. S'ensuit la conception dans laquelle est donnée la structure mise en place pour prendre en charge cette fonctionnalité et finalement les tests unitaires mis en place sur les différentes classes du module.

4.1 Analyse

La géo-localisation telle qu'elle a été faite dans [14] permet d'obtenir des informations sur une adresse IP routable grâce à la méthode **HashTable geoLocalisationIP (InetAdress ip)**. Plusieurs ressources sont utilisées par cette méthode :

- Une adresse IP routable à analyser
- Une base de données pour la géo-localisation contenue dans un fichier
- Un fichier de configuration définissant différents paramètres pour la base de données indiquant l'intervalle temporel de mise à jour de la base de données et l'emplacement du fichier de données

Globalement la méthode **geoLocalisationIP** utilise la base de données contenue dans le fichier GeoliteCity.dat pour géo-localiser une adresse IP. Cette base de données doit être mise à jour à intervalles réguliers. L'intervalle de temps de mise à jour est définit dans le fichier de configuration ainsi que l'emplacement du fichier de données.

La gestion de ces ressources doit être prise en compte dans le module de géo-localisation. Normalement ces ressources devaient être gérées directement en interne dans le travail [14]. Mais la façon dont c'est fait ne convient pas aux spécifications de l'analyseur modulaire. En effet la spécification 1.3.1.24 implique que la mise à jour de la base de données ne ralentisse pas la méthode de géo-localisation. Or cette mise à jour est faite directement dans la méthode ce qui peut ralentir considérablement son temps de réponse. Il faut donc

gérer le chargement et la mise à jour de la base de données pour l'externaliser de la méthode **geoLocalisationIP**. Un *thread* de gestion est donc mis en place pour gérer ces mises à jour. Son fonctionnement et l'implémentation du module sont détaillés dans la section suivante.

4.2 Conception

L'utilisation de la géo-localisation faite dans [14] implique l'utilisation de la classe «`UA_IPSource_IPLocIP`» qui fournit la méthode d'obtention d'information de géo-localisation pour une adresse IP. Mais aussi et surtout, l'utilisation de la classe «`GeoLocation`» qui gère les données de géo-localisation. Ci-après se trouve un schéma de classe simplifié décrivant le module et les classes de «`UA_IPSource_IPLocIP`» qui sont adaptées pour faire ce que l'on veut.

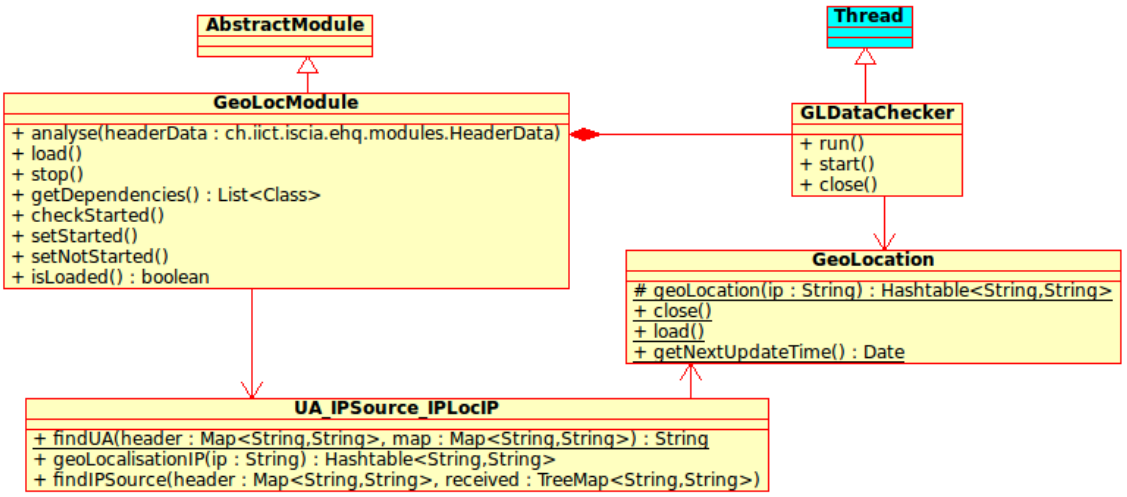


FIGURE 4.1 – Diagramme de classe simplifié du module de géo-localisation

On peut voir le point d'entrée du module «`GeoLocModule`» qui utilise «`UA_IPSource_IPLocIP`» pour pouvoir géo-localiser l'adresse IP publique de la machine émettrice potentiellement présente dans «`HeaderData`». On voit aussi que «`GeoLocModule`» utilise un *thread* «`GLDataChecker`» pour gérer les mises à jour des données utilisées par «`UA_IPSource_IPLocIP`». En fait le *thread* utilise directement la classe «`GeoLocation`» qui gère les données de géo-localisation utilisées par «`UA_IPSource_IPLocIP`». Ci après sont donnés un bref descriptif de chaque classes.

4.2.1 classe GeoLocModule

Cette classe étends «`AbstractModule`» pour donner le point d'entrée au gestionnaire de module ce qui permet de mettre en place les fonctionnalités nécessaires à sa gestion.
Sa méthode **getDependencies** doit retourner une liste contenant la classe «`IPAnalyserModule`» pour que le gestionnaire de modules puisse traiter les dépendances correctement.
«`GeoLocModule`» possède en attribut statique un *thread* de gestion des données nommé «`GLDataChecker`» qui s'occupe de gérer les mises à jour. Il est détaillé ci-dessous. Ce

thread est créé et initialisé durant le chargement du module (méthode **start**). Il est arrêté à la fermeture du module (méthode **close**). En dehors de ces deux méthodes le *thread* est indépendant de «`GeoLocModule`». La classe représentant ce *tread* est détaillée ci-après.

4.2.2 classe GLDataChecker

Cette classe étends la classe *thread*. Elle a été créée pour palier au problème de la mise à jour en parallèle qui n'était pas fait par défaut. Le diagramme d'activité ci-après montre le fonctionnement du *thread*.

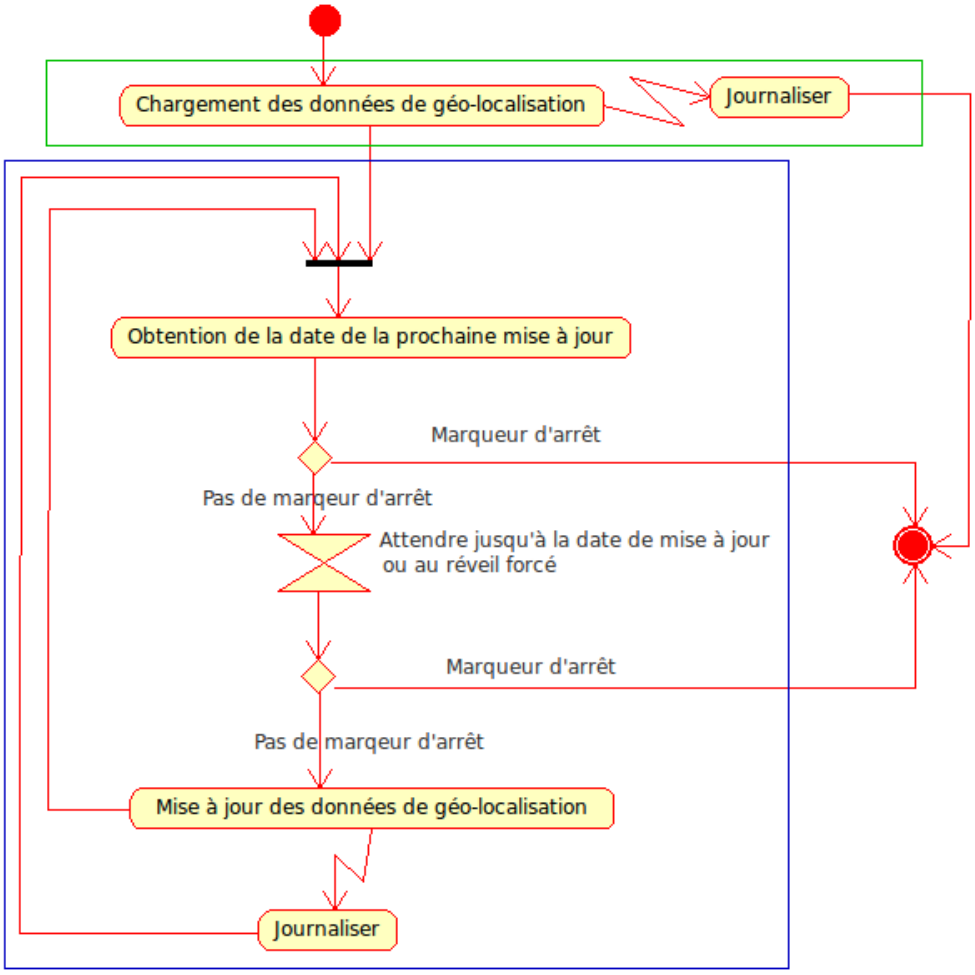

Chargement des données de géo-localisation
Journaliser
Obtention de la date de la prochaine mise à jour
Marqueur d'arrêt
Pas de marqeur d'arrêt
Attendre jusqu'à la date de mise à jour
ou au réveil forcé
Marqueur d'arrêt
Pas de marqeur d'arrêt
Mise à jour des données de géo-localisation
Journaliser

FIGURE 4.2 – Diagramme d'activité du *thread* de mise à jour des données de géo-localisation

L'encadré vert représente le traitement de la méthode **start** et le rouge celui de la méthode **run**. La méthode **close**, non représentée sur ce schéma permets d'activer le marqueur d'arrêt et de réveiller le *thread* si il est en attente.
Le chargement des données, la mise à jour des données et l'obtention de la date de mise à jour se fait en utilisant la classe «`GeoLocation`» qui est détaillée dans la suite du document.

4.2.3 Classe UA_IPSource_IPLocIP

Cette classe est utilisée telle-qu'elle est fournie dans [14]. On peut voir trois méthodes dans cette classe. Cependant le module n'utilise que la méthode **geoLocalisationIP** pour géo-localiser les adresses IP. Cette méthode retourne les arguments suivants :
- countryCode
- countryName
- regionName
- regionCode
- zipcode
- cityName
- latitude
- longitude
- metroCode
- area_code
- dmaCode
- timeZone

Ces arguments sont en suite insérés par GeoLocModule dans le HeaderData avec les marqueurs correspondants donnés dans le cahier des charges.

4.2.4 Classe GeoLocation

Cette classe est la classe d'origine utilisée par «`UA_IPSource_IPLocIP`». Elle est modifiée pour externaliser la mise à jour des données de géo-localisation. Le travail fait sur cette classe est simple. Il a suffit de séparer les bouts de codes concernant la lecture du fichier de configuration et la mise à jour de la base de données et du fichier dans la méthode **load** créée à cet effet. Cette méthode permet de faire le traitement "chargement des données" et "mise à jour des données" et fait les actions suivantes :

1. Lecture du fichier de configuration
2. Calcul de la date de mise à jour
3. Mise à jour si nécessaire
4. Calcul de la prochaine date de mise à jour

La technique de calcul de mise à jour et la technique de mise à jour sont laissées telle qu'à l'origine.

En plus de la méthode **load**, une méthode **close** a été mise en place. Le but étant de pouvoir fermer de façon propre le gestionnaire utilisé pour aller lire dans la base de données lors de l'arrêt du module. (cette méthode est appelée via la méthode **close** de «`GLDataChecker`»)

4.2.5 Fichier de configuration

La gestion de la configuration du module est gérée d'origine dans le travail [14]. Elle n'est donc pas modifiée. Sa description est reprise ci-dessous :

```
<configurationGeoLocation>
```

```
        <update>1m</update>
        <pathGeoLocation>GeoLiteCity.dat</pathGeoLocation>
</configurationGeoLocation>
```

Le champ "update" permet de définir l'intervalle de mise à jour. Cet intervalle peut être donné en mois (m) ou en jours (d) précédé d'un entier.
Le champ "pathGeoLocation" permet de renseiger l'emplacement du fichier de données. Cet emplacement

Ce fichier ce trouve dans le fichier EmailHeaderQualif.jar disponible sur le DVD dans l'annexe E et aussi dans le répertoire de projet «src/main/java/ressource» sur ce même DVD.

Ceci clos la partie d'analyse du module de géo-localisation. La suite du document traite de la réalisation de celui-ci.

4.3 Réalisation

La réalisation de cette classe n'a posé qu'un seul problème : l'emplacement des fichiers de ressources utilisés pour la géo-localisation. Le fichier de configuration et le fichier de données doivent être à des emplacement spécifiques qui doivent être mis dans le code source. Ceci étant, Maven permet de définir un dossier de ressources [1] que le projet utilise.

Ayant utilisé cette possibilité pour garantir une intégration facile dans *ISCIA*, le problème suivant s'est posé :

> Lors de la génération du fichier JAR final, toutes les ressources sont inclues dans le fichier JAR. Ceci étant, elles ne sont plus accessibles via la classe «`File`» qui ne peut aller lire les fichiers se trouvant dans une archive JAR.

La solution à ce problème est d'utiliser la méthode **InputStream getResourceAsStream (String name)** [2] (source : [36]) qui permets d'obtenir un «`InputStream`» pour pouvoir lire des fichiers de ressource. Cette méthode est valable pour n'importe quel fichier. Qu'il soit dans le système de fichier ou dans le fichier JAR. À ceci près que la chaîne de caractère du nom de fichier doit posséder un "/" en premier caractère pour les fichiers qui se trouvent dans le JAR.

Cette technique permet de lire un fichier sans soucis. Ce qui convient très bien pour le fichier de configuration, mais n'est pas suffisant pour le fichier de données car il est impossible d'écrire avec cette façon de faire. Le fichier de données doit donc se trouver à l'extérieur du fichier JAR et, in extendo, à l'extérieur du dossier de projet qui contient les ressources.

La section suivante décrit les tests effectués sur ce module.

1. On entends par dossier de ressources un dossier contenant les fichiers et dossiers requis pour le projet
2. Cette méthode est fournie par la classe «`Class`»

4.4 Tests

Les tests unitaires de ce modules sont ceux qui ont été faits dans [14]. Ils se trouvent dans le répertoire de projet «src/test/JUnitsTests».

Des tests unitaires supplémentaires ont été faits pour la classe «`GeoLocModule`». C'est le fichier de test «GeoLocModuleTest.java» qui se trouve dans le paquetage «ch.iict.iscia.ehq.modules.geoloc» à l'emplacement «src/test/java» qui le fait.

Globalement y sont testés le démarrage et l'arrêt du module tels que définit dans la figure 4.2 ce qui inclut le contrôle du marquer d'arrêt.

Le prochain chapitre traite du module d'obtention de l'agent utilisateur utilisé pour envoyer un courriel.

Chapitre 5

Module User Agent

Ce chapitre traite de l'intégration de l'obtention de l'agent utilisateur faites par R. Cornu dans [14] chapitre 4 dans un module nommé UserAgent. Premièrement une analyse de ce qui est fournit et de la façon de l'intégrer est faite. Puis la partie "réalisation" donne des informations sur la manière dont a été réalisé ce module et finalement une brève description des tests faits sur les classes du module est donnée.

5.1 Analyse

Comme dis dans le chapitre précédent la classe «`UA_IPSource_IPLocIP`» fournit les différentes méthodes faites dans [14] et notamment la méthode **String findUA (Map<String, String> header, Map<String, String> firstReceived)**. Cette méthode permet d'obtenir l'agent utilisateur contenu dans un en-tête de courriel passé en premier paramètre. Le deuxième paramètre est le premier *received* qui se trouve dans l'en-tête (le plus vieux). Il ne cera pas traité ici de la façon dont «`UA_IPSource_IPLocIP`» obtient ces informations car c'est un travail qui a déjà été fait dans [14] mais uniquement des points de son intégration tels que les ressources utilisées et la structure à mettre en place pour l'intégrer.

Pour pouvoir faire son travail, cette méthode utilise, indirectement, des fichiers XML définissant des règles de détection d'agent utilisateur. Plus précisément, les fichiers «FullTrace.xml» et «PartialTrace.xml» de plus un fichier de configuration «configFindUA.xml» est aussi utilisé.

La gestion de ces fichiers se passe directement durant l'appel de la méthode **findUA**. La méthode va charger les conifguration lors de son premier appel. Puis, lors des appels suivants, elle va recharger ces fichiers si un certain laps de temps (définit dans le fichier de configuration) c'est écoulé depuis le dernier chargement. Cette façon de gérer les changements de fichier fait que ça ne corresponds pas à la spécification 1.3.1.24.

Il est clair que le chargement de fichiers XML ne pose pas en soi une grande augmentation du temps de réponse de la méthode. Mais cela rend la méthode plus sensible aux erreurs liées à ces fichiers. En externalisant ces chargements de la méthode, il est possible de diminuer les risques d'erreurs. Par exemple :

Un des fichiers de configuration est manquant. À chaque exécution de la méthode **findUA** une exception sera levée alors que le module est démarré.

Si la gestion des données est externalisée au module, cette erreur ne peut survenir puisque le module n'aurai pas pu être démarré sans les fichiers de configuration.

Tout comme pour le module de géo-localisation, le plus simple est de mettre en place un *thread* de gestion des données dans le module. Il résulte de ce raisonnement et des prérequis d'un module la structure suivante :

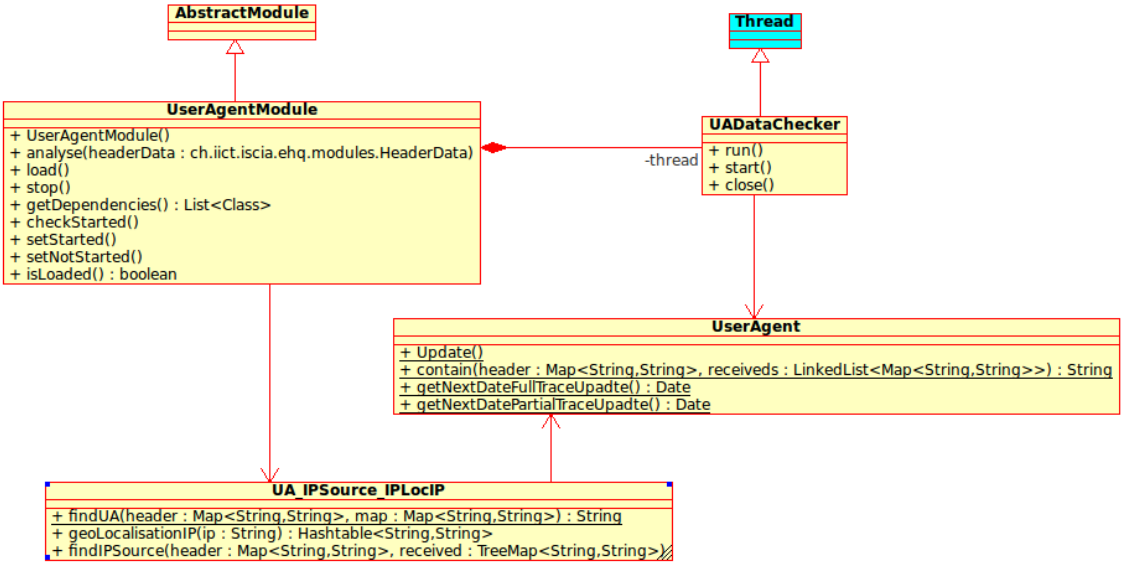


FIGURE 5.1 – Diagramme de classe simplifié du module user agent

La classe «`UserAgentModule`» donne le point d'entrée du module en étendant la classe AbstractModule. Sa méthode **analyse** utilise la méthode **findUA** de «`UA_IPSource_IPLocIP`» pour pouvoir obtenir l'agent utilisateur à partir de l'en-tête qui lui est fournit. «`UserAgentModule`» possède un *thread* de mise à jour «`UADataChecker`» définit en statique pour éviter les *threads* de mise à jour concurrents. Ce *thread* va utiliser la classe «`UserAgent`» pour gérer les mises à jour. Une description plus détaillée de ces classe ce trouve dans la suite de ce document.

5.1.1 Classe UserAgentModule

Cette classe étends «`AbstractModule`» pour donner le point d'entrée au gestionnaire de module ce qui permet de mettre en place les fonctionnalités nécessaires à sa gestion.

Sa méthode**getDependencies** retourne une liste vide car ce module ne dépends d'aucun autre.

«`UserAgentModule`» possède en attribut statique un *thread* de gestion des données nommé «`UADataChecker`» qui s'occupe de gérer les mises à jour. Il est détaillé ci-dessous. Ce *thread*

est créé et initialisé durant le chargement du module (méthode **start**). Il est arrêté à la fermeture du module (méthode **close**). En dehors de ces deux méthodes le *thread* est indépendant de «`UserAgentModule`». La classe représentant ce *tread* est détaillée ci-après.

5.1.2 Classe UADataChecker

La gestion de la mise à jour des données par cette classe est reprise du module de géo-localisation. Cette classe étends donc la classe Java™ «`Thread`». Ci dessous se trouve le diagramme d'activité de gestion des mises à jour.

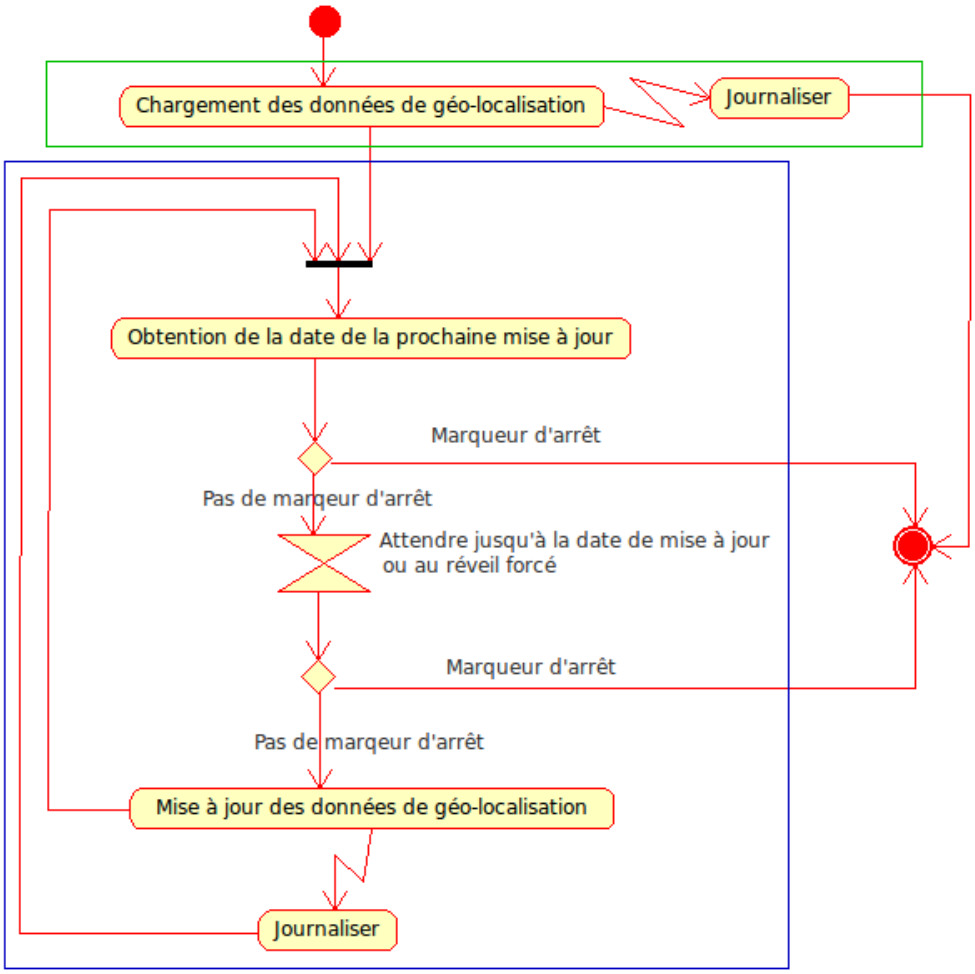

Chargement des données de géo-localisation
Journaliser
Obtention de la date de la prochaine mise à jour
Marqueur d'arrêt
Pas de marqeur d'arrêt
Attendre jusqu'à la date de mise à jour
ou au réveil forcé
Marqueur d'arrêt
Pas de marqeur d'arrêt
Mise à jour des données de géo-localisation
Journaliser

FIGURE 5.2 – Diagramme d'activité du *thread* de mise à jour des données de UserAgent

L'encadré vert représente le traitement de la méthode **start** et le rouge celui de la méthode **run**. La méthode **close**, non représentée sur ce schéma permets d'activer le marqueur d'arrêt et de réveiller le *thread* si il est en attente.

Le chargement des données, la mise à jour des données et l'obtention de la date de mise à jour se fait en utilisant la classe «`UserAgent`» qui est détaillée dans la suite du document.

5.1.3 Classe UA_IPSource_IPLocIP

Comme dis précédemment, seule la méthode **findUA** est utilisée dans cette classe faite dans [14]. Le seul point important à relever est que cette méthode utilise «`UserAgent`» pour trouver l'agent utilisateur. C'est donc «`UserAgent`» qui est concerné pour la mise à jour des données.

5.1.4 Classe UserAgent

Tout comme pour la classe «`GeoLocation`» citée dans le chapitre précédent, cette classe est reprise de [14]. Elle a été adaptée pour externaliser la mise à jour des fichiers XML de sa méthode **contain** qui fait l'obtention de l'agent utilisateur.

Pour adapter cette classe, il a suffit de séparer les bouts de codes concernant la lecture du fichier de configuration et la mise à jour de la base de données et du fichier dans la méthode **load** créée à cet effet. Cette méthode permet de faire le traitement "chargement des données" et "mise à jour des données" et fait les actions suivantes :

1. Lecture du fichier de configuration
2. Calcul de la date de mise à jour
3. Mise à jour si nécessaire
4. Calcul de la prochaine date de mise à jour

La technique de calcul de mise à jour et la technique de mise à jour sont laissées telle qu'à l'origine.

5.1.5 Fichiers de configuration

La structure des fichiers «FullTrace.xml» et «PartialTrace.xml» qui contiennent des règles de détection ne sera pas discutée ici. Voir [14] pour plus d'informations sur ces deux fichiers. En revanche le fichier de configuration «configFindUA.xml» possède la structure suivante :

```
<configurationPath>
        <pathFullTrace>/FullTrace.xml</pathFullTrace>
        <pathPartialTrace>/PartialTrace.xml</pathPartialTrace>
        <update>
                1m
        </update>
</configurationPath>
```

Il s'y trouve les paramètres "pathFullTrace" et "pathPartialTrace" qui donnent les chemins des fichiers de règle de détection.

Le champ "update" permet de définir l'intervalle de mise à jour. Cet intervalle peut être donné en mois (m) ou en jours (d) précédé d'un entier.

L'analyse du module ce termine ici. La suite du document traite de la réalisation et des tests du module.

5.2 Réalisation

La seule particularité durant la réalisation est la même que pour le module de géo-localisation. Ce sont les ressources qui ont posé problème. Une fois ce problème résolu, le module user agent a fonctionné sans problèmes.

5.3 Tests

Les tests unitaires de ce modules sont ceux qui ont été faits dans [14]. Ils se trouvent dans le répertoire de projet «src/test/JUnitsTests».

Des tests unitaires supplémentaires ont été faits pour la classe «`UserAgentModule`». C'est le fichier de test «UserAgentModuleTest.java» qui se trouve dans le paquetage «ch.iict.iscia.ehq.modules.userage à l'emplacement «src/test/java» qui le fait.

Globalement y sont testés le démarrage et l'arrêt du module tels que définit dans la figure 4.2 ce qui inclut le contrôle du marquer d'arrêt.

Le prochain chapitre traite du module Whois qui permet d'obtenir l'intervalle d'une adresse IP publique et son propriétaire.

Chapitre 6

Module Whois

Selon la [37] le protocole Whois permet à un client de faire une requête sur un serveur en format texte sur le port 43. Le serveur répond alors avec du texte. Le but initial de ce protocole est de pouvoir avoir des informations contenues dans une base de données. Ainsi, la commande Whois n'est à l'origine qu'un moyen d'obtenir un renseignement par un serveur sur un texte envoyé (voir [6] pour plus d'informations).

La commande «Whois » s'est adaptée à l'ouverture d'internet et permet à présent d'obtenir des informations sur une adresse IP ou sur un nom DNS. Cette commande est donc toute désignée pour fournir des informations sur des adresses IP trouvées dans des en-têtes de courriels. Le module dont on parle ici a été appelé Whois car c'est le but qu'il cherche à atteindre en obtenant des informations sur les adresses IP publiques trouvées par le module «IP Vérification».

Dans ce chapitre sont données les spécifications du module. C'est à dire, les contraintes auxquelles doit répondre le module. Puis vient la conception où est expliquée la façon dont s'intègre le module dans EmailHeaderQualif et la façon dont il est implémenté. La réalisation donne des informations sur le codage du module et des adaptations qui ont dû être apportées à la conception et finalement les tests donnent les résultats et commentaires sur les tests préalablement définis dans les spécifications.

6.1 Spécification

Ici sont expliquées les fonctionnalités et contraintes du module Whois . Premièrement il est important de voir que ce module a besoin d'une adresse IP publique pour travailler. Il est donc dépendant du module IPAnalyser (voir chap. 3 pour plus d'informations sur ce module).

Le module Whois doit deuxièmement répondre aux spécifications de l'API le concernant. Ces différentes spécifications, combinées aux contraintes définies dans 2.1, ont des conséquences sur le module. Ces conséquences sont listées ci-après :

– Trouver, si possible, l'intervalle d'une adresse IP publique de la machine émettrice d'un courriel et l'associer, si possible au nom de l'organisation à laquelle elle appartient.

- Ne pas augmenter exagérément le temps de réponse de EmailHeaderQualif en cherchant ces informations.
- Avoir un fichier de données que l'on charge au lancement du module. Si les données ne sont pas chargeables, le module n'est pas démarré et une exception est levée.
- Une exception est levée lors de l'appel de la méthode d'analyse si le module n'a pas été démarré.
- Mettre à jour périodiquement les données selon des paramètres prédéfinis dans un fichier de configuration sans ralentir le processus de traitement de EmailHeaderQualif , ce qui implique l'utilisation de *threads* pour paralléliser les mises à jour (voir [21] pour plus de détails sur les *threads*).
- Le ou les *threads* du module ne doivent pas posséder de priorité haute ou être en temps réel pour ne pas monopoliser le processeur.
- Toutes les informations d'exécution du module doivent passer par le système de journalisation log4j, ce qui inclut les exceptions, les erreurs pouvant survenir, le démarrage et l'arrêt du module ainsi que celui de son/ses *threads* de mise à jour.
- La mise à jour périodique des données est également journalisée ainsi que sa réussite ou son échec.
- Mettre à disposition des méthodes d'arrêt et de démarrage du module qui doivent comprendre le démarrage et l'arrêt des *threads* gérés par le module.
- Déclarer la dépendance au module "IP Verification" pour avoir l'adresse IP publique à analyser.
- Si l'adresse IP publique n'est pas disponible, simplement quitter la méthode d'analyse sans autre forme de traitement ou de journalisation.
- Toutes les informations de configuration nécessaires au module se trouvent dans un fichier de configuration. Notamment l'emplacement du fichier de données et l'intervalle de mise à jour de celui-ci.

Ce sont les différentes contraintes et conséquences qui peuvent être sorties des spécifications.

La section suivante donne la conception découlant de ces contraintes et en prenant compte de la conception de l'analyseur modulaire.

6.2 Conception

Ici sont traités les différents points soulevés dans la spécification pour en faire une structure de code et de fonctionnement pour le module. Puis est donnée la structure du module et son fonctionnement. Ensuite est détaillée la méthode de stockage et d'accès des données et la gestion de la mise à jour des données.

Le but de ce module est donc d'obtenir un intervalle d'adresses IP et son propriétaire en fonction de l'adresse IP publique fournie.

Premièrement, il faut prendre en compte que ce module possède le cycle de vie défini dans 2.2.3.2 avec les deux états “démarré” et “arrêté”. Deuxièmement, pour pouvoir s'intégrer correctement à EmailHeaderQualif le module doit posséder une classe qui étende la classe

«`AbstractModule`» (c.f. 2.2.3.3). C'est le point d'entrée du module.

Le module doit disposer d'une adresse IP publique à analyser. C'est l'adresse publique qui est extraite par «IP Verification» ce qui implique une dépendance sur ce module pour «Whois ».

Pour pouvoir obtenir des informations, il est clair qu'il serait possible d'exécuter une requête whois sur un serveur et d'en interpréter le résultat. Cependant, comme défini dans les spécifications, le temps de réponse maximal à une requête sur EmailHeaderQualif est court. Mais le temps de réponse à une requête whois sur un serveur est très variable et peut augmenter significativement le temps de réponse de EmailHeaderQualif . En effet, en dehors du temps de réponse standard, il est possible que la connexion internet soit coupée ou encore lente. Cette méthode n'est donc pas acceptable. Pour faire les choses correctement, il faut posséder des données locales qui permettront d'obtenir les informations sur les adresses IP. La structure de ces données est détaillée dans 6.2.2.

Puisqu'il faut utiliser des données locales, il faut aussi les mettre à jour. En effet, les attributions d'adresses IP et d'intervalles à des organismes sont sujettes à changements.

L'utilisation de données locales et les mises à jour impliquent qu'il faut les charger en mémoire, utiliser un fichier de données pour les stoker et ne pas ralentir EmailHeaderQualif lors de la mise à jour de celles-ci. Le plus simple pour résoudre tous ces prérequis est de mettre en place un système de *threads* qui permet de charger en mémoire et mettre à jour les données régulièrement.Le *thread* possède le fonctionnement décrit dans le diagramme d'activité qui suit.

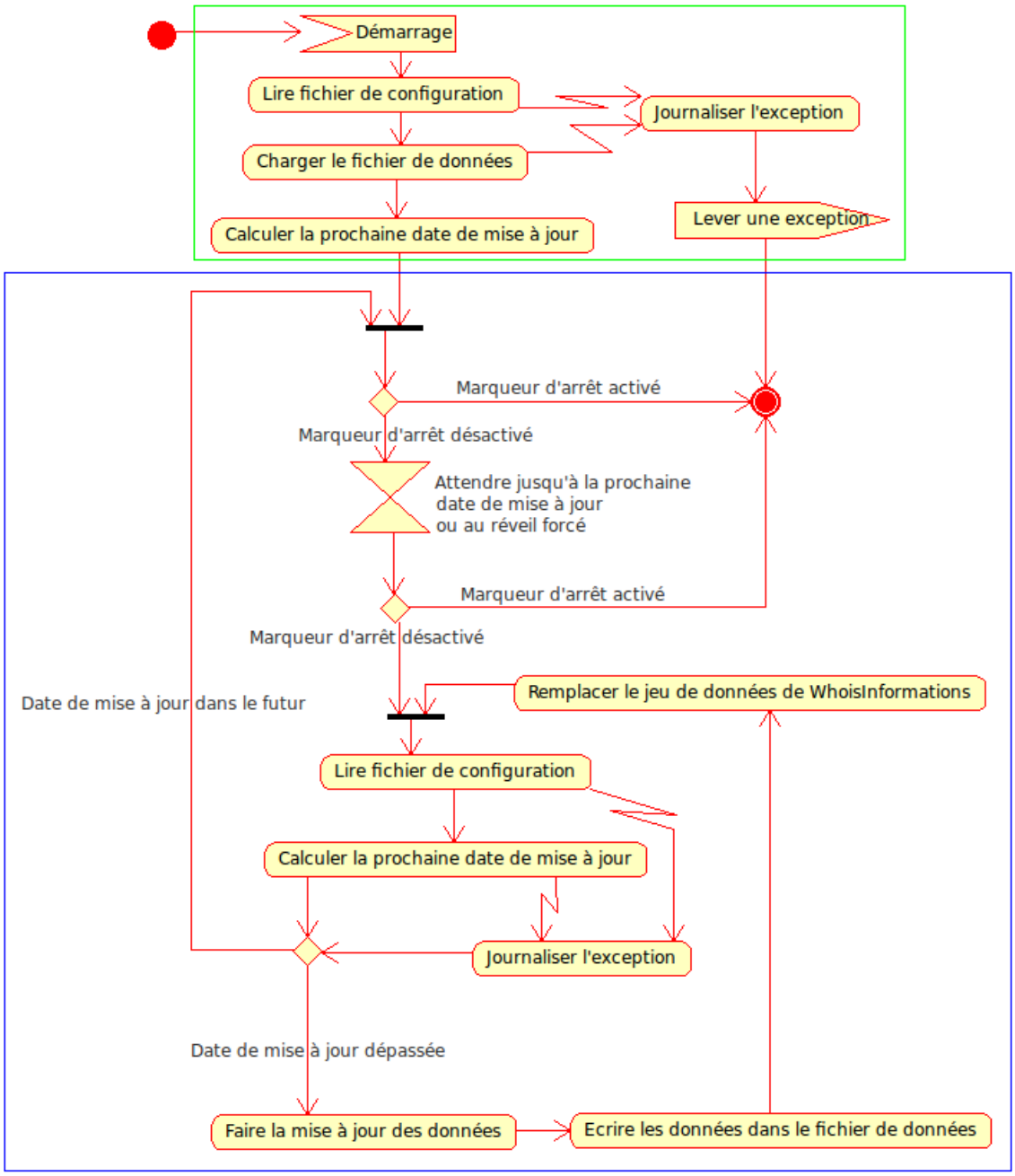

Démarrage
Lire fichier de configuration
Journaliser l'exception
Charger le fichier de données
Lever une exception
Calculer la prochaine date de mise à jour
Marqueur d'arrêt activé
Marqueur d'arrêt désactivé
Attendre jusqu'à la prochaine
date de mise à jour
ou au réveil forcé
Marqueur d'arrêt activé
Marqueur d'arrêt désactivé
Date de mise à jour dans le futur
Remplacer le jeu de données de WhoisInformations
Lire fichier de configuration
Calculer la prochaine date de mise à jour
Journaliser l'exception
Date de mise à jour dépassée
Faire la mise à jour des données
Ecrire les données dans le fichier de données

FIGURE 6.1 – Diagramme d'activité du *thread* de mise à jour du module Whois

Premièrement, il y a deux cadres de couleur. Le cadre de couleur verte correspond à la méthode **start** du *thread*, soit l'initialisation du *thread*. Le cadre bleu correspond à l'exécution du *thread* soit la méthode **run**.

Deuxièmement on constate par deux fois la présence du contrôle d'un marqueur d'arrêt. C'est tout simplement un booléen qui permet d'arrêter le *thread* de mise à jour. Ce booléen

est mis à «vrai» lors de l'appel de la méthode **close** non représentée sur ce schéma. Cette méthode réveille aussi le *thread* au cas où il serait en attente dans la méthode run ce qui permet de terminer à coup sûr le *thread*.

La méthode **start** possède le fonctionnement défini dans l'encadré vert de la figure 6.1.

La méthode **run** possède le fonctionnement défini dans l'encadré bleu de la figure 6.1.

Le calcul de la date de mise à jour se base sur la date de modification du fichier de données. En considérant que le fichier a été modifié au temps t la prochaine mise à jour est faite au temps t + I où I représente l'intervalle de mise à jour défini dans le fichier de configuration.

La suite du document donne la structure du module découlant de cette analyse.

6.2.1 Structure de base du module

Des points levés précédemment découlent les points suivants :
- Une classe doit étendre «`AbstractModule`» pour donner une interface au contrôleur de modules avec le module Whois . Cette classe s'appelle «`WhoisInformations`».
- Une classe doit étendre la classe «`Thread`» pour pouvoir gérer les mises à jour. C'est la classe «`WhoisRetriever`».
- Une classe doit être créée pour contenir les données à gérer et à utiliser. C'est la classe «`WhoisData`».
- La classe «`WhoisData`» contient plusieurs entrées contenant des intervalles d'adresses IP et des données associées. Ce sont les données trouvées grâce à «`WhoisRetriever`» ou qui étaient dans le fichier de données. Un intervalle d'adresses IP est représenté par la classe «`IPRange`» qui contient deux adresses IP.
- La classe «`IP`» permet de représenter une adresse IPv4 (voir [1]) (On ne travaille pas avec les adresses IPv6).
- Une classe «`WhoisDataFile`» doit être créée pour pouvoir lire et écrire les données «`WhoisData`» dans un fichier. Elle étend la classe «`File`» pour hériter de toutes les méthodes que l'ont peut désirer pour un fichier et met à disposition une méthode de lecture de fichier et une méthode d'écriture de fichier.
- Une classe permettant d'analyser le fichier de configuration c'est la classe «`ParseConfigurationFileWhois`» Elle étend «`DefaultHandler`» afin de pouvoir facilement utiliser le XMLReader fournit par Java™ qui permet de lire le contenu d'un fichier XML (Pour plus de détail sur le XML voir [22]).

Le schéma de classe simplifié ci-dessous représente la structure du module. Un détail de chaque classe se trouve à la suite du schéma.

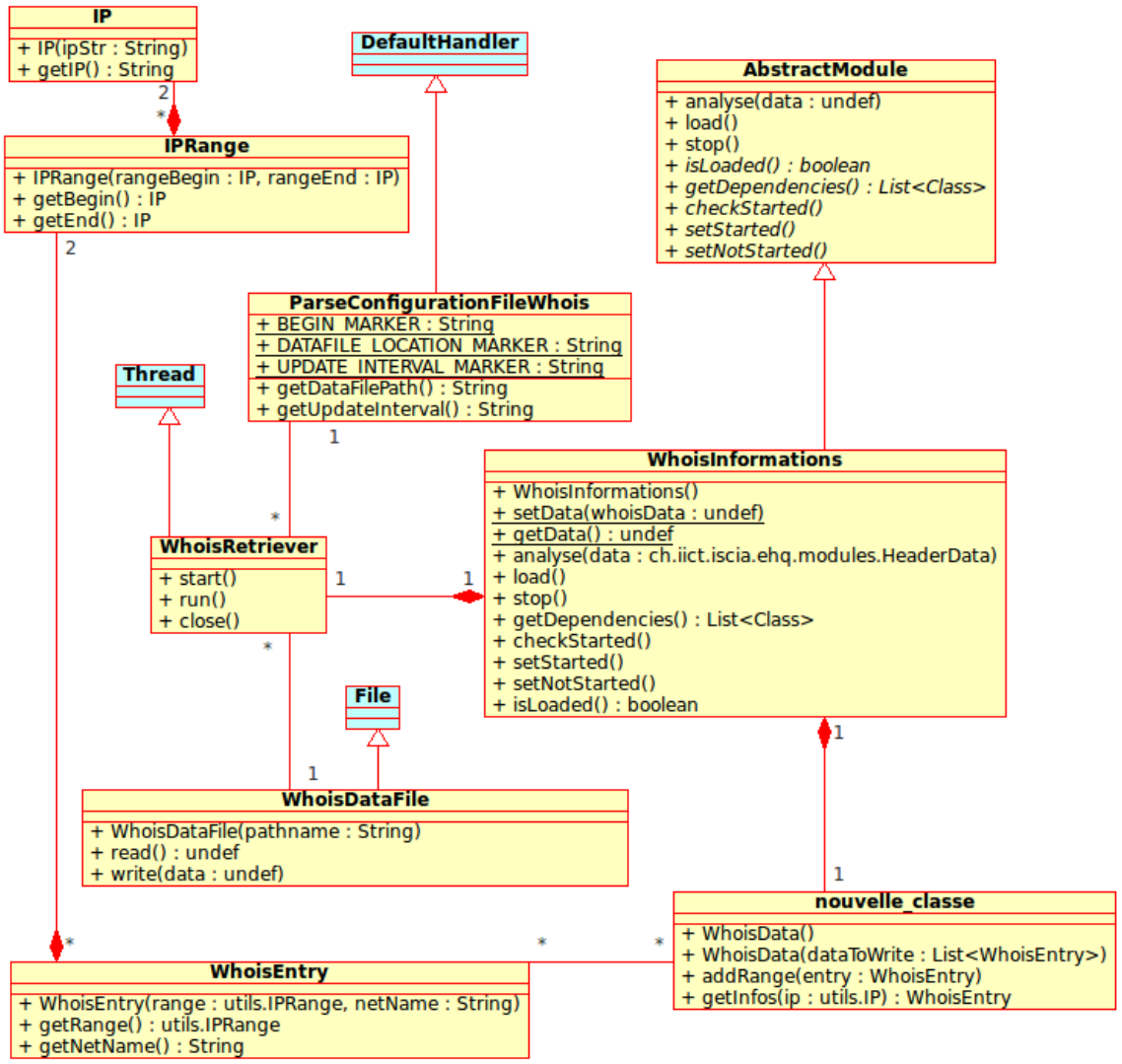


FIGURE 6.2 – Diagramme de classe simplifié du module Whois

Les classe en bleu sont des classes fournies parJava™.

6.2.1.1 WhoisInformations

Cette classe est la classe d'entrée du module pour l'analyseur modulaire. Son but est de pouvoir donner l'intervalle de l'adresse IP publique d'une machine émettrice contenue dans les données passées en paramètre et le propriétaire de cet intervalle. Elle permet d'initialiser les ressources utilisées par le module et de les libérer (le *thread* de mise à jour fait

partie de ces ressources). Elle étend «`AbstractModule`» (voir 2.2.3.3) pour pouvoir être utilisée par le contrôleur de modules de EmailHeaderQualif .
De plus cette classe défini deux méthodes supplémentaires à «`AbstractModule`» accessibles uniquement dans le paquetage de la classe (*protected*) :

- *setData*
- *getData*

Ces deux méthodes permettent à «`WhoisRetriever`» de pouvoir, respectivement, modifier/obtenir les données utilisées par «`WhoisInformations`» dans sa méthode **analyse**. Il est important de prendre en compte que les données «`WhoisData`» contenues dans la classe sont en code statique. Il n'y a donc qu'un seul jeu de données pour toutes les instances de la classe. Ceci premièrement parce que la classe n'est pas censée être instanciée plusieurs fois. Mais aussi pour des raisons d'optimisation (c.f. 2.3.1). Il en va de même pour le *thread* de mise à jour qui est défini en statique dans la classe. Ceci permet de limiter les erreurs pouvant survenir à cause de deux *threads* de mise à jour s'exécutant en parallèle.
La méthode **analyse** de la classe va juste chercher l'adresse IP publique fournie dans le conteneur data qui lui est passé en paramètre et va vérifier dans ses données «`WhoisData`» si il n'y a pas un intervalle et un propriétaire correspondant. Si un intervalle est trouvé, il est rajouté dans les données du conteneur ainsi que son propriétaire.

6.2.1.2 WhoisRetriever

Ce *thread* permet de gérer les données. C'est à dire qu'il s'occupe d'aller lire le fichier de configuration du module en utilisant «`ParseConfigurationFileWhois`» (détaillé ci après), puis de charger les données à partir du fichier dont le chemin est défini dans le fichier de configuration. Et finalement s'occupe de mettre à jour les données à intervalle régulier. Cet intervalle est aussi défini dans le fichier de configuration (voir 6.2.4 pour plus d'informations sur le fichier de configuration). Ci-après sont décrites les différentes actions effectuées par les méthodes publiques du *thread* :

1. **start** effectue les actions définie dans l'encadré vert de la figure 6.1 page 62
2. **run** effectue les actions définie dans l'encadré bleu de la figure 6.1 page 62
3. **close** Permet d'arrêter le *thread* (sortie de la méthode run) en mettant le marqueur d'arrêt et en réveillant le *thread* pour qu'il se termine.

6.2.1.3 WhoisDataFile

Pour pouvoir lire et écrire dans un fichier les données utilisées, il faut soit mettre en forme ces données sous forme de texte pour les écrire dans un fichier et les ré-extraire au moyen d'un parseur de texte. Ou alors, comme dans le cas de «`WhoisDataFile`», écrire des données sérialisables [23] dans un fichier de données.

Cette classe étend la classe «`File`» pour lui apporter tous ses avantages (gestion des fichiers ,etc.). C'est un apport facile et qui ne coûte qu'une extension de classe. Cependant, deux méthodes supplémentaires ont été définies :

read Retourne les données («`WhoisData`») contenues dans le fichier représenté par cette classes

write Écrit les données «`WhoisData`» passées en paramètre dans le fichier représenté par cette classe. Les données qui étaient déjà dans le fichier sont écrasées. Si il n'existe pas il est créé.

La sérialisation de «`WhoisData`» implique que la classe implémente l'interface «`Serializable`» qui est un marqueur de sérialisation pour Java™. Toutes les données non statiques de «`WhoisData`» doivent implémenter cette interface et ceci de façon récursive pour ces classes.

Avantages de la sérialisation des données
- Facile à mettre en place.
- Pas d'erreur dans l'écriture ou le parsing contrairement à un fichier texte.
- La modification d'une classe est prise en compte pour la lecture et l'écriture directement sans avoir à modifier tout le système d'écriture/lecture (parseur etc.).

Inconvénients de la sérialisation des données
- Avoir un fichier de base est obligatoire dans le module. Si le fichier est perdu il faut en recréer un.
- Il est impossible de lire un fichier de données si les attributs, le paquetage ou le nom des classes sérialisées ont changé.
- Empêche la portabilité à d'autres langage.

Raison du choix de la sérialisation La sérialisation a été choisie car elle est facile et rapide a mettre en place et évite les erreurs dues au parsing des données lors de la lecture. De plus les modifications sur les classes sérialisées sont prise en compte sans autre modification. Il n'a pas été pris le XML pour les raisons inverses.

6.2.1.4 IP

Cette classe est une classe simple qui permet de représenter une adresse IP. Elle n'utilise qu'un constructeur qui prend en paramètre un «`String`» représentant une adresse IP et permet de récupérer cette adresse IP avec la méthode **getIP** sous forme de string.

La classe Java™ «`InetAddress`» n'est pas utilisée, car il est très simple de représenter une adresse IP de 32 bits avec un *long*. Ce qui la rend aisément comparable et est utile dans ce projet (voir 6.2.2 pour plus de détails). Mais encore cette classe est une source d'erreurs possibles. En effet, si on essaye de créer une adresse IP à partir d'une chaîne de caractère

non valide, on risque d'obtenir des effets de bords inattendus puisque cette chaine peut être interprétée comme un nom DNS. La classe va alors tenter de résoudre ce nom et risque soit de provoquer une erreur soit de fausser les données.

Une adresse IP n'est pas modifiable car elle n'a pas de setter.

6.2.1.5 IPRange

Le but de cette classe est simplement de représenter les différents intervalles contenus dans «`WhoisData`». Elle prend donc en paramètre les deux adresses IP qui représentent son début et sa fin. La première adresse IP donnée dans le constructeur doit être plus petite que la deuxième, sans quoi une exception est levée.

Un intervalle n'est pas modifiable, ceci pour représenter le fait que ce qu'il représente est obtenu par une requête whois et que ce résultat ne change pas.

6.2.1.6 WhoisEntry

Cette classe permet de représenter une entrée dans «`WhoisData`». Si on fait le parallèle avec une requête whois elle peut représenter un fragment de son résultat. Elle contient un «`IPRange`» et le nom de l'organisation à laquelle cet intervalle appartient (nommé par la suite *NetName* ou propriétaire). Son constructeur prend en paramètre un «`IPRange`» et un String représentant le propriétaire. Les méthodes **getNetName** et **getRange** permettent de les récupérer. Il est volontaire de ne pas pouvoir modifier les données de cette classe car le seul moyen d'obtenir des informations sur un intervalle d'adresses IP pour le module est la mise à jour par le «`WhoisRetriever`». Et vu que la mise à jour des données se base sur un jeu de données «`WhoisData`» vide (voir 6.2.2.3), il ne va jamais modifier une entrée existante.

6.2.1.7 WhoisData

La classe représente un jeu d'entrées whois («`WhoisEntry`»). Elle a pour but de pouvoir faire une recherche sur une adresse IP et d'en ressortir la «`WhoisEntry`» correspondante avec la méthode **getInfos**.

Son constructeur peut prendre en paramètre une liste d'entrées, ceci dans le but de faciliter les tests unitaires mais aussi pour donner la possibilité au *thread* de mise à jour de créer facilement un jeu d'entrées, si besoin est.

La méthode **addRange** permet de rajouter une entrée. La suppression d'une entrée n'est pas prise en compte. Premièrement, car la mise à jour par le *thread* se base sur un nouveau jeu de données (c.f. 6.2.2.3) et deuxièmement parce que c'est lors de l'insertion de données que l'on va gérer les différents problèmes qu'il pourrait y avoir dans le jeu de données.

La structure de données qui est utilisée pour gérer les entrées et la façon dont elle est gérée est détaillée dans la suite de ce document.

6.2.2 Gestion des données

La gestion des données utilisées par le module se sépare en deux parties : le stockage et la gestion. Le stockage, détaillé ci-après, traite de la structure de données utilisée et la gestion des conflits du comportement de cette structure lors de l'ajout et de la recherche.

6.2.2.1 Stockage des données

La classe «`WhoisData`» doit gérer des entrées de données. Il est important de comprendre ce que représente une de ces entrées. Fondamentalement c'est le résultat d'une requête d'informations sur une adresse IP à un serveur. Mais en remontant plus haut il faut voir que c'est une information sur la répartition des adresses IP dans le monde.
En considérant qu'un intervalle d'adresses IP est attribué uniquement à un seul organisme, on comprend qu'il ne peut pas y avoir d'intervalles qui se chevauchent. C'est à dire que deux intervalles ne peuvent jamais avoir d'adresse IP en commun. Il est donc possible d'ordonner ces intervalles et de les utiliser comme clé de tri pour le stockage des entrées et la recherche de résultats.

Cependant, au moins une exception existe. En effet en faisant une requête whois sur un espace d'adressage non alloué (0.0.0.0 par exemple) par *IANA*, on obtient l'intervalle le plus grand possible, soit de l'adresse 0.0.0.0 à l'adresse 255.255.255.255. Il recouvre donc tous les autres. Si on considère qu'il existe au moins une exception, il peut en exister plusieurs. Par exemple ARIN qui est chargé de l'attribution de l'espace d'adressage pour la partie Amérique du Nord peut très bien retourner l'intervalle entier dont il est responsable si on fait une requête sur une adresse qui ne fait partie d'aucun intervalle attribué. Mais ce genre de donnée n'apporte aucune informations pertinente dans le cadre de ISCIA. Les intervalles englobants seront donc simplement ignorés.

Mais encore, il faut prendre en compte que la quantité de données à gérer est très grande. En effet, en considérant qu'il y a 32 bits pour une adresse réseau soit $2^{32} - 1$ adresses possibles, en considérant que le dernier byte représente un sous-réseau il reste $2^{24} - 1$ sous-réseaux possibles. Évidemment, en considérant les sous-réseaux privés et les spéciaux, il n'y a pas autant de sous-réseaux. Mais la quantité d'intervalles possibles reste tout de même conséquente.

En considérant que chaque intervalle est unique et peut être ordonnancé, on arrive à la conclusion qu'un arbre binaire est le meilleur choix possible. Il possède un temps d'ajout et d'obtention des informations en un temps Olog(N) (c.f.[16] p.202). La structure de données à mettre en place est donc un arbre.

Cependant, aucune classe par défaut de Java™ ne représente directement une structure de données en arbre. Mais la «`TreeMap`» utilise cette structure. Elle permet d'associer une clé avec une valeur dans un nœud de l'arbre en se basant sur la clé pour placer le nœud. En considérant une entrée «`WhoisEntry`» comme la valeur et l'intervalle qu'il contient comme

clé, la structure de données ainsi obtenue convient parfaitement à nos besoins. La seule contrainte de cette structure est de rendre la clé comparable. La classe «`IPRange`» doit donc implémenter l'interface comparable.

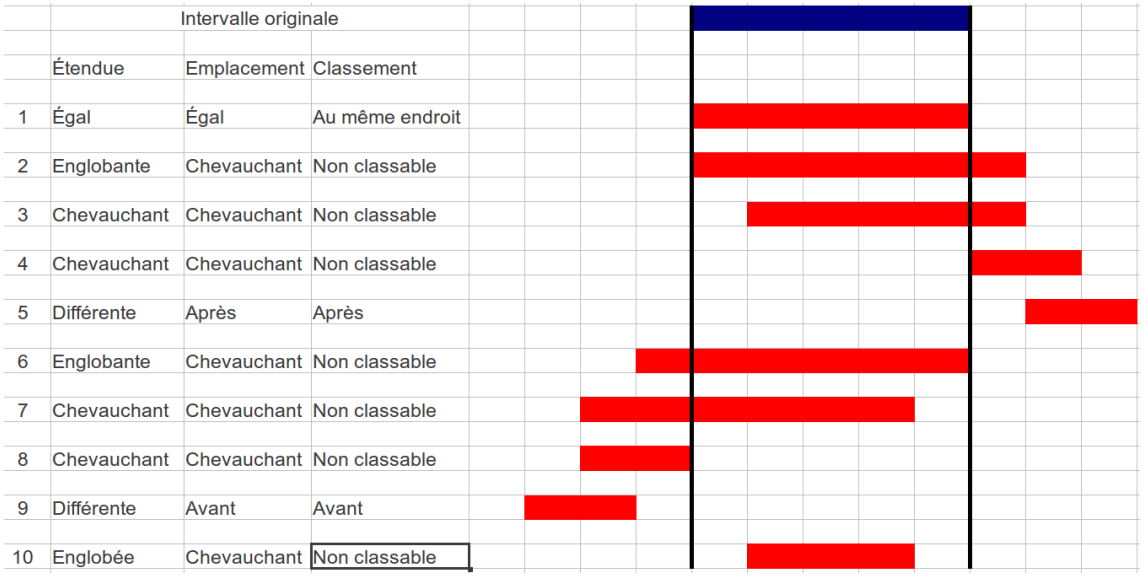
Le «`TreeMap`» permet d'obtenir une entrée à partir de n'importe quel objet comparable (c.f. [18]). Pour obtenir une entrée à partir d'une adresse IP, il suffit donc de rendre la classe «`IP`» comparable en implémentant l'interface Comparable de Java™. Le détail du fonctionnement de la comparaison est donné dans la suite de ce document.

L'ajout de données dans un «`TreeMap`» ne se fait pas sans risques. Les risques et les solutions trouvées sont donnés dans la suite du document. Comme dit précédemment la suppression de données n'est pas prise en compte.

6.2.2.2 Ajout et recherche de données dans un jeu existant

Les données sur les différents intervalles d'adresses IP sont difficiles à gérer. Par exemple l'ajout d'un intervalle chevauchant un intervalle déjà inséré dans la liste des données pose problème. Et l'ajout d'un intervalle contenu dans un autre intervalle pose également problème. Certes ces deux cas ne devraient théoriquement pas survenir. Mais cela peut arriver. Ne serait-ce que par le risque de changement des données sur les serveurs durant la mise à jour. Mais en plus, comme dit en 6.2.2.1, il faut rendre les intervalles comparables et les adresses IP aussi. Il faut donc définir une politique de traitement pour l'ajout et la recherche de données. C'est la façon dont on implémente l'interface Comparable qui va entrer en jeu. Mais il faut aussi modifier le comportement par défaut de la «`TreeMap`» comme il est expliqué dans la suite.

Premièrement voici les différentes possibilités de comparaison entre deux intervalles :



	Étendue	Emplacement	Classement
1	Égal	Égal	Au même endroit
2	Englobante	Chevauchant	Non classable
3	Chevauchant	Chevauchant	Non classable
4	Chevauchant	Chevauchant	Non classable
5	Différente	Après	Après
6	Englobante	Chevauchant	Non classable
7	Chevauchant	Chevauchant	Non classable
8	Chevauchant	Chevauchant	Non classable
9	Différente	Avant	Avant
10	Englobée	Chevauchant	Non classable

FIGURE 6.3 – Comparaison d'intervalles

Cette figure s'interprète comme suit :
- L'intervalle d'adresses IP original est celle en bleu
- Les deux traits noirs descendants représentent les adresses IP de début et de fin de l'intervalle original
- Chacune des barres rouges propose une possibilité de comparaison d'intervalle avec l'intervalle original
- La colonne étendue donne une comparaison entre les deux étendues d'adresses (quelles sont les adresses couvertes)
- La colonne Emplacement indique l'emplacement de l'intervalle par rapport aux possibilités d'adresses IP
- La colonne classement donne le résultat de classement entre les deux intervalles (emplacement de l'intervalle rouge par rapport à l'intervalle bleu).

Les intervalles qui sont «non classables», le sont soit parce qu'ils sont chevauchant (adresses en commun), soit parce qu'ils sont englobés. Le cas englobant est le même que l'englobé mais inversé. Le résultat de la comparaison est donc identique (non classable).

La comparaison telle qu'elle est utilisée dans Java™ [17] retourne un entier. Si on compare un élément A à un élément B (A.compareTo(B) en Java™). Les résultats possibles sont les suivants :

- Plus petit que 0 si B est placé avant A
- Égal à 0 si leur emplacement est identique
- Plus grand que 0 si B est plus grand que A

Dans ce cadre que faire des cas «non classables» qui pourraient survenir lors d'une insertion ? Trois options sont à disposition :

- Ignorer et ne pas insérer
- Lever une exception
- Remplacer les données

L'ajout de données se passe toujours durant une mise à jour. La mise à jour ne prenant pas compte des données pré-existante avant celle-ci, on peut considérer que les cas non classables sont des cas exceptionnels et qu'il suffit de les ignorer puisqu'il y a visiblement une incohérence. Attention toutefois le cas de l'intervalle englobant et englobé n'entre pas dans cette considération pour des raisons de précision de données. Ceci est expliqué dans la suite.

Le but de ce module est de renvoyer des données précises mais aussi, indirectement, de donner les informations requises pour un maximum d'adresses IP. Si on veut garder cette précision il convient de prendre en compte les intervalles englobants. La figure suivante montre le comportement lors de l'insertion d'un intervalle englobant.

FIGURE 6.4 – Gestion des intervalles englobants

Les intervalles bleus représentent les intervalles déjà présents et l'intervalle vert, l'intervalle à ajouter. On constate que l'intervalle vert est réparti dans les espaces vides qui ne sont pas couverts par les autres intervalles. Ceci pour garder un maximum de précision dans les données «`WhoisData`».

En prenant le cas inverse : l'ajout des intervalles bleu alors que l'intervalle vert est déjà présent, le résultat est le même. Ceci pour donner une précision maximale aux données.

Pour éviter de retourner des données comme le IANA-BLOCK cité précédemment, les intervalles «xxx.0.0.0-xxx.255.255.255» sont filtrés lors de l'insertion pour ne pas être ajoutés. Ces intervalles n'apporte rien d'un point de vue de EmailHeaderQualif ou pour une investigation.
Ce constat implique de modifier le comportement de la méthode *put(key, value)* de la «`TreeMap`» pour adapter l'insertion à ces spécifications lors de l'insertion d'un intervalle dans «`WhoisData`».

Dans le cas d'une égalité on considère que les données ajoutées sont plus récentes que les données originales. On remplace donc les données originales.

6.2.2.2.1 Recherche de données La recherche de données relative à une adresse IP est simple à mettre en place. Comme dit précédemment, il suffit de rendre comparable «`IP`» à un intervalle. Il faut donc implémenter l'interface correspondante. Les résultats de la méthode de comparaison entre une adresse IP A et un intervalle B (A.compareTo(B) en Java™) sont les suivants :

- Plus petit que 0 si l'adresse de début de l'intervalle est plus grande que A
- Égal à 0 si A est compris dans B
- Plus grand que 0 si A est plus grand que la dernière adresse de B

Ainsi la méthode *get(Object o)* de la «`TreeMap`» retournera le «`WhoisEntry`» dont l'intervalle comparée avec A donne 0. Si aucun intervalle ne correspond *null* est retourné.

Maintenant que la structure et la gestion des données whois sont définis il faut poser la méthode de mise à jour des données. C'est ce qui est traité à la suite.

6.2.2.3 Méthode de mise à jour des données

La mise à jour des données est l'un des points critiques de ce module. En effet, l'obtention des données est compliquée. Une requête whois passe par des serveurs pour trouver les différentes informations. Les serveurs utilisés par défaut sous Ubuntu sont les 5 serveurs fournis par les organismes du registre Internet régional (RIR voir [27]) : RIPE-NCC, ARIN, APNIC, LACNIC et AfriNIC, soit respectivement : whois.ripe.net, whois.arin.net, whois.apnic.net, whois.lacnic.net et whois.afrinic.net.

Dans l'idéal il faudrait utiliser des bases de données fournies par ces organismes. Cependant, seul RIPE fournit la possibilité de télécharger un dump de cette base de données. ARIN, APNIC, LACNIC et AfriNIC n'offrent pas directement cette possibilité. Cependant, un code source Java™ trouvé sur `http://www.java2s.com/Open-Source/Java-Document/Search-Engine/yacy/de/anomic/net/whois.java.htm` permet de faire une requête Whois en utilisant les ressources système. Ce code permet, après quelques tests, de faire correctement des requêtes Whois . Après un nombre conséquent de tests à la suite (2000), il apparaît que les requêtes ne sont pas bloquées par les serveurs.

Cependant, pour des raisons de portabilité et de sécurité, ce code ne sera pas pris mais le code trouvé sur [28] est un code source sous licence GPL[19]. Il fournit de base un moyen de faire un whois sur des noms DNS mais peut aussi être adapté pour le faire sur des adresses IP.

L'algorithme de mise à jour des données est représenté dans le diagramme d'activité suivant. C'est la partie «faire la mise à jour des données» du diagramme 6.1 :

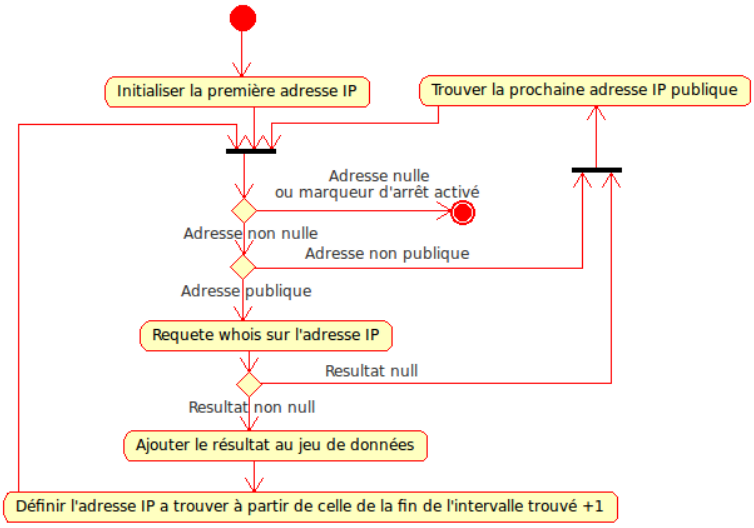

Initialiser la première adresse IP
Trouver la prochaine adresse IP publique
Adresse nulle
ou marqueur d'arrêt activé
Adresse non nulle
Adresse non publique
Adresse publique
Requete whois sur l'adresse IP
Resultat null
Resultat non null
Ajouter le résultat au jeu de données
Définir l'adresse IP a trouver à partir de celle de la fin de l'intervalle trouvé +1

FIGURE 6.5 – Diagramme d'activité de mise à jour des données whois

La suite du document traite de la procédure de mise à jour par le *thread* «`WhoisRetriever`».

6.2.3 Fonctionnement du *thread* de mise à jour des données

Comme dit dans 6.2.1.2 le *thread* de mise à jour des données possède trois méthodes. Elles ont les interactions suivantes en fonction de l'état [20] du *thread* :
État NEW

– La méthode **start()** a le comportement défini dans l'encadré vert de la figure 6.1 page 62

État RUNNABLE, BLOCKED, WAITING, TIMEDWAITING

– La méthode **stop()** Arrêter immédiatement le *thread* en mettant le marqueur d'arrêt à vrai et en réveillant le thread.
– La méthode **run()** ne doit pas pas être appelée mais son code est le cœur de la mise à jour par le *thread*. Ses actions sont définies dans l'encadré bleu de la figure 6.1 page 62.

Toutes les combinaisons État-Méthode qui ne sont pas définies ici n'ont aucune incidence.

Les données déjà existantes dans «`WhoisInformations`» ne sont pas réutilisées. Premièrement pour éviter des erreurs. Mais surtout parce qu'une mise à jour couvre tout l'intervalle d'adresse IP possible. Il n'est donc pas nécessaire de faire cela.

La prochaine section donne des informations relatives à la possibilité de paralléliser la mise à jour.

6.2.4 Fichier de configuration

Le fichier de configuration du module possède les informations suivantes :

- Chemin du fichier de données
- Intervalle de mise à jour

Le chemin du fichier de données peut être mis en relatif ou absolu. Cependant, il ne doit pas être dans le fichier JAR de EmailHeaderQualif .

L'intervalle de mise à jour peut être définie en mois, jours, heures, minutes. L'option minutes a été prise pour faciliter les tests. Le format pour indiquer l'intervalle est xT où x est un entier et T est égal à 'm', 'd', 'h', 'min' pour respectivement mois, jours, heures, minutes. Voici un exemple de fichier de configuration :

```
<configurationWhoisModule>
        <dataFileLocation>whoisData.dat</dataFileLocation>
        <updateIntervall>5m</updateIntervall>
</configurationWhoisModule>
```

Ce choix de structure s'inspire du travail [14].

Ce fichier ce trouve dans le fichier EmailHeaderQualif.jar disponible sur le DVD dans l'annexe E et aussi dans le répertoire de projet «src/main/java/ressource» sur ce même DVD.
L'analyse et la spécification du module ont été faites. La section suivante traite la réalisation de ce module.

6.3 Réalisation

Dans ce chapitre sont donnés les points importants révélés durant la réalisation. C'est l'adaptation du code source trouvé et la gestion des réponses infructueuses lors de requêtes whois.

6.3.1 Adaptation du code source trouvé

Le code source trouvé sur [28] a dû être adapté. Son code source ne se trouve pas dans le même répertoire paquetage que le restant du projet vu qu'il a été repris et par respect pour son auteur. Il se trouve dans le répertoire «src/main/java/org/neverfear/whois» de l'annexe E. Sa structure est la suivante :

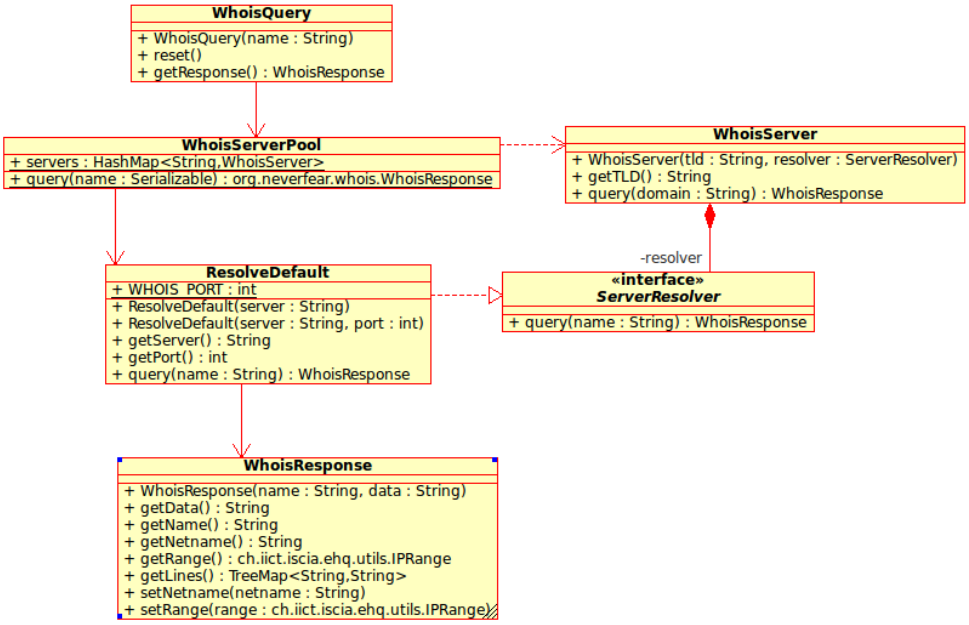


FIGURE 6.6 – Diagramme de classe simplifié du code source «Whois» utilisé

Pour faire une requête sur un serveur "Whois", il suffit de créer une «`WhoisQuery`» et d'exécuter son code **getResponse**. Cette méthode va appeler la méthode **query** de «`whoisServerPool`» qui, en fonction du nom passé en paramètre, va aller consulter la liste de serveurs définie en statique pour faire la requête. Le serveur en question utilise un résolveur pour exécuter sa requête («`ResolveDefault`») qui va retourner un objet «`WhoisResponse`» comme réponse.

Trois adaptations sont apportées à ce code :

- WhoisResponse a été adapté avec deux attributs supplémentaires : Le netName et et l'intervalle
- Une classe par RIR[35] étends la classe «`ResolveDefault`» et redéfinit la méthode **query** pour remplir le «`WhoisResponse`» retourné avec le NetName et l'intervalle. Une classe par RIR est nécessaire car chacun formule les réponses aux requêtes à sa manière.
- La liste des serveurs contenue dans «`WhoisServerPool`» a été complétée pour que les requêtes sur les adresses IP se fassent via le bon résolveur en fonction du premier byte de l'adresse IP. Par exemple, une entrée avec la chaine de caractère "2" est définie avec la classe «`ResolveRipe`» pour que toutes les requêtes faites sur les adresses IP "2.x.x.x" utilisent le résolveur RIPE qui se chargera de compléter la «`WhoisResponse`».

6.4 Limitation du nombre de requêtes «whois» faites en cas de réponses nulle

Le temps de réponse d'une requête «whois» est d'à peu près 3 secondes. Mais le temps de réponse d'une requête whois qui n'aboutit pas peut être de plus de 20 secondes (ce temps dépends du timeout définit qui a été mis à 20 secondes). En partant de ce point et en consi-

dérant le diagramme 6.5 page 74, la partie «trouver la prochaine adresse IP publique» de ce diagramme fait incrémenter la recherche de une adresse IP à chaque fois qu'une requête n'a pas aboutit. Ce qui fait que si, par exemple, un serveur tel que «whois.arin.net» ne réponds pas, toutes les adresses IP d'un intervalle tel que «2.0.0.0-2.255.255.255» sont testées. Le résultat est que 2^24 requêtes sont faites dans le vide. Si chaque requête prends encore 20 secondes, le temps de mise à jour des données Whois est augmenté de 64 jours rien que pour cet intervalle.

Pour résoudre ce problème rencontré, le nombre de requêtes consécutives est contrôlé. Si il dépasse deux requêtes infructueuses, le troisième byte de l'adresse IP est incrémenté de 1. Puis on recommence le contrôle de deux requêtes infructueuses et c'est alors le deuxième byte qui est incrémenté de 1. Si deux requêtes infructueuses se font encore, on passe au prochain intervalle général en incrémentant le premier byte et en mettant tout les autres à 0.

Le temps ainsi perdu pour un intervalle pour lequel il n'y a pas de réponses est de 2^2*20 secondes soit 1 minute et 20 secondes au lieu des 64 jours. Ainsi, il est aussi contrôlé que c'est tout l'intervalle pour lequel il n'y a pas de réponse et non pas les quelques premières adresses.
La suite de ce chapitre traite des tests effectués sur ce module.

6.5 Tests

Deux catégories de tests sont prévus pour ce module : les tests unitaires et les tests de validation qui sont définis dans les spécifications (chapitre 1.3). Les tests de validation sont utilisés pour contrôler le bon fonctionnement du module durant son développement. Dans la suite de cette section sont donnés les explications sur ces trois catégories.

6.5.1 Tests Unitaires

Afin de contrôler le code fait et/ou modifié, des tests unitaires permettant de contrôler certaines méthodes fournies par les diverses classes sont mis en place. La liste des classes testées ce trouve ci-après. Le code source de ces tests unitaires se trouve dans le répertoire «src/test/java/ch/iict/iscia/ehq/module/whois» et de E «src/test/java/ch/iict/iscia/ehq/utils».

«`WhoisDataFile`» Il est contrôlé que cette classe écrit et lit correctement les données «`WhoisData`».

«`WhoisData`» Pour cette classe, il est contrôlé que l'ajout se comporte tel que définit en 6.2.2.

«`WhoisInformations`» Le contrôle du démarrage et de l'arrêt de module est fait dans ses tests unitaires. Il y est aussi contrôlé qu'avec un jeu de donnée «`WhoisData`» les bons résultats sont retournés par la méthode **analyse**.

«`WhoisRetriever`» Aucun tests unitaires n'a été fait pour cette classe. Le test des mises à jour à été fait à la main en utilisant les cas-tests de validation et en contrôlant que le fichier de données était bien mis à jour quand il était périmé lors du lancement d'une mise à jour.

«IPRange» La comparaison entre deux intervalles telle que définie dans 6.2.2 est contrôlée ainsi que le constructeur qui ne doit pas pouvoir prendre de paramètre *null* sans lever une exception. Le constructeur doit aussi prendre deux adresses IP ordonnées (la deuxième doit être plus grande que la petite).
«IP» La comparaison de deux adresses IP et d'une adresse IP avec un intervalle tels que définis dans 6.2.2 sont testés. La création d'une adresse IP avec un format incorrect est contrôlé et l'obtention d'une adresse IP à partir d'une autre adresse que l'on incrémente est contrôlé (notamment le comportement aux bornes 0.0.0.0 et 255.255.255.255).

Le chapitre Whois se termine ici. Le chapitre suivant traite du module IPBL.

Chapitre 7

Module IPBL

Le module IPBL n'est pas implémenté en raison du temps à disposition. Il a été décidé que cette partie serait donnée pour un prochain travail de diplôme car la quantité de travail qu'elle comporte est très conséquente. Cependant une structure de base a été mise en place pour pouvoir intégrer facilement ce futur travail dans les modules. Elle est décrite dans ce chapitre en suivant la structure d'analyse, conception, réalisation et Tests.

7.1 Analyse

Comme tout les autres modules, celui-ci doit posséder une classe servant de point d'entrée pour le gestionnaire de modules. C'est ce qui a été mis en place ici. Il n'y a donc qu'une seule classe à créer qui s'appelle «`IPBLCheckModule`» pour faire le point d'entrée du module. Le restant du travail étant à faire.
Le but de ce module est de travailler sur une adresse IP publique de machine source il posséde donc une dépendance sur le module IP Verification.
Il est important de prendre en compte qu'un tel module devra posséder des données et qu'il faudra les gérer. Pour ce faire il est conseillé de s'inspirer du thread de mise à jour fait dans de module Whois définit dans le chapitre 6.

7.2 Conception

La structure du module est simple car c'est la structure de base de tout modules :

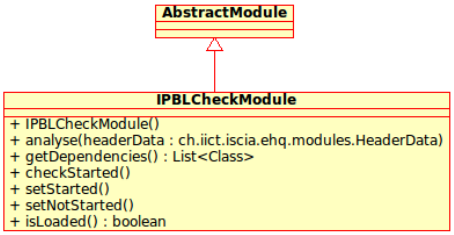


Figure 7.1 – Diagramme de classe du module IPBL

La seule partie codée dans «`IPBLCheckModule`» est la gestion du démarrage et de l'arrêt de ce module et la déclaration de dépendance au module IP Verification. Le restant du code doit être fait dans un futur travail.

7.3 Réalisations

La réalisation de ce module n'a rien de spécifique puisque elle n'est composé que d'une seule classe. De plus le code de cette classe a été fait à partir de celui d'autres modules codés précédemment.

7.4 Tests

Un simple test unitaire est mis en place. Il permet de contrôler si le module démarre et s'arrête correctement et si la dépendance à IP Verification est bien déclarée. C'est le fichier «IPBLCheckModuleTest» dans le répertoire «src/test/java/ch/iict/iscia/ehq/modules/ipbl» de l'annexe E qui contient ces tests.
Ce chapitre clos la partie «développement» de ce projet. Le chapitre suivant traite de la validation faite sur ce projet.

Chapitre 8

Validation

Dans ce chapitre sont traités les résultats des deux validations effectuées durant le projet et leurs conséquences. Les tableaux de résultats finaux de validation se trouvent dans l'annexe A

8.1 Première validation

La première validation a fait ressortir le besoin de posséder un moyen de tester EmailHeaderQualif avec des fichiers texte contenant des courriels complets. En effet la version originale du programme de test obligeait la création des fichiers de données pour les tests à partir de fichier texte pré-formatés à la main. Une fois ces fichiers de données créés il était possible de les reprendre pour pouvoir faire les tests désirés.

Finalement pour faire un test sur un courriel donné, il fallait :

1. Créer un fichier texte avec uniquement l'en-tête du courriel à analyser
2. Mettre en forme cet en-tête à la main
3. Utiliser le programme de test pour créer un fichier de test
4. Lancer le test sur le fichier de test

Ces étapes sont fastidieuses et apportent peu au projet. En revanche, en modifiant le programme de test pour prendre des fichiers texte bruts contenant des courriels (un courriel par fichier), il a été créé un outil simple de test mais aussi d'évaluation de jeu de données puisque le programme de test est capable d'aller analyser tous les fichiers contenus dans un répertoire et d'en ressortir des statistiques.

Ces modifications ont été possibles grâce au mandant qui a fournit un code permettant d'extraire les informations à partir de fichier texte qu'il a suffit d'intégrer. En revanche il a fallu corriger tout les fichiers de tests d'origine pour correspondre au nouveau programme de test ce qui a pris un certain temps.

Mise à part le point du programme de test modifié, le restant des modifications concerne les cas-tests eux-même. Il a fallu corriger les cas-test pour les rendre plus pratique pour la

personne les faisant. C'est à dire qu'il a fallu réunir des cas test et rendre plus explicite et précis leur description et leur exécution.

Pour ce qui est des fonctionnalités à tester tout était OK par rapport au code déjà fait.

8.2 Deuxième validation

La deuxième validation a fait ressortir plusieurs points :

- Enlever la description complète des exceptions levées dans la journalisation
- Ajouter la date dans les fichiers de journalisation
- Problème de bannissement d'IP par les serveurs whois de RIPE et de ARIN

Le dernier point est très important pour l'intégration de ce projet. En effet au moment de la deuxième validation, le temps imparti pour corriger ce problème n'était plus suffisant pour le résoudre. C'est donc avant son intégration qu'il faut corriger ce point. Fort heureusement, par aquis de conscience, tous les tests impliquant le module whois étaient fait depuis le domicile. Ce qui fait que l'adresse IP publique de la HEIG-VD n'a pas été bloquée par ces serveurs. Lors de la pré-étude il avait été établi qu'il était possible de faire des requêtes sans limitation sur les 5 serveurs whois des RIR. Il apparaît cependant que ce n'est pas le cas.

Suite à ce problème, les différentes solutions possibles pour résoudre ce problème ont été étudiées. La solution trouvée est de limiter le nombre de requête sur une journée. En effet, après étude des différentes politique de gestion des requêtes sur les sites officiels de *RIPE* [31], AfriNIC [30], APNIC [32] et ARIN [33], il est confirmé que, théoriquement, il n'y a aucune limite au nombre de requête qui peuvent être faites sur leurs serveurs. Cependant pour des raisons de qualité de service chacun de ces organisme mets en place un système de comptage du nombre de requêtes par jour pour pouvoir bloquer les attaques de déni de service. Le nombre de requêtes tolérées par tranche de 24 heures n'est malheureusement pas diffusé par ces organismes.

Il faut donc limiter le nombre de requêtes par jour pour la mise à jour des données.

Le chapitre suivant revient sur le déroulement du projet.

Chapitre 9

Retour sur le déroulement du projet

Dans ce chapitre est donné un bref retour sur le déroulement du projet. Les documents complets retraçant le déroulement du travail sont disponibles dans l'annexe E dans le répertoire «suivi».

L'annexe B montre un tableau décrivant le suivi des heures de travail du projet. On peut voir dans cette annexe que le nombre d'heures totales de travail dépasse largement le nombre d'heures prévues. Les raisons de ce dépassement sont données ci-après.

Dans la phase initiale du projet, beaucoup de temps a été utilisé pour définir le plus précisément possible le cahier des charges et la planification correspondante. Initialement, cette tâche devait être effectuée durant les deux premières semaines du projet. Mais cela à pris plus de temps que prévu suite aux multiples corrections et entrevues avec le professeur responsable. Que ce soit pour des raisons de calendrier ou de précision, cette définition à débordé sur la première semaine de développement du projet. De plus la correction de ce cahier des charges impliquait des recherches ce qui a augmenté le retard pris initialement.

Le développement du gestionnaire de modules et l'adaptation des modules tirés de [14] a pris moins de temps que prévu dans l'ensemble. Mais les multiples corrections des spécifications ont augmenté le temps de travail dévolu à l'analyseur modulaire ce qui a impacté sur le début de la tâche du module Whois.

Ce retard ajouté à la sous-estimation du temps nécessaire pour développer le module Whois a impliqué l'abandon du module IPBL et la prolongation du projet d'une semaine.

Mise à part ces points concernant le retard, le projet s'est bien déroulé. Tout au long du projet les multiples entretiens avec le professeur responsable ont permis de garder le travail sur une ligne directrice et une méthodologie de développement correcte.

Ce commentaire sur le déroulement du travail clos le corps de ce document. Dans le chapitre suivant est donné une conclusion qui revient sur l'ensemble du projet.

Chapitre 10

Conclusion

Ce chapitre finalise le rapport en revenant sur les objectifs initiaux et les performances vis-à-vis de ces objectifs. Il s'y trouve aussi une section sur l'intégration de EmailHeaderQualif dans un autre projet et la valeur ajoutée que cette API apporte à ISCIA. Ensuite le déroulement et les difficultés techniques rencontrées durant le projet sont expliqués. Finalement les apports personnels du projet closent ce document.

10.1 Retour sur les objectifs initiaux

En résumé les objectifs initiaux étaient :

1. Identifier l'adresse IP de la source d'un courriel
2. Identifier le premier MTA utilisé pour transmettre un courriel
3. Intégrer le travail de Romain Cornu [14]
4. Obtenir l'intervalle, et son propriétaire, de l'adresse IP source d'un courriel
5. Qualifier l'adresse IP source d'un courriel et l'adresse IP du premier MTA utilisé pour transmettre le courriel si le temps le permets

6. Créer un système de traitement modulaire

Tout les points ci-dessus sont réalisés excepté la qualification d'adresse IP pour laquelle le temps imparti na pas été suffisant.
Un des objectifs était de pouvoir créer facilement des nouveaux modules venant s'ajouter facilement au traitement fait par EmailHeaderQualif . C'est chose faite. Pour ce faire il suffit de créer comme point d'entrée du module une classe qui étende «`ch.iict.iscia.ehq.modules.AbstractModu`
la compiler et la placer dans le dossier contenant le code java exécutable correspondant. En rajoutant le nom complet de la classe (incluant le paquetage) dans le fichier de configuration «configModuleControler» et en redémarrant EmailHeaderQualif le nouveau module sera pris en compte.
Les performances de EmailHeaderQualif sont données dans la section suivante.

10.1.1 Performances

Une fois le projet terminé des statistiques de performance de EmailHeaderQualif ont été effectuées sur un jeu de courriels fournit par le mandant. Ce jeu de données est fournit sous forme de fichiers texte représentant des courriels. Ces fichiers se trouvent dans l'annexe E dans le répertoire «fichiers_test/TestAll».

La taille de l'échantillon est de 35262 courriels. Dans ces courriels deux d'entre eux sont sans *received*. Ces deux cas ne représente donc que 0.01% du jeu de données. Leur impacte est donc négligeable.

Le graphe suivant représente le pourcentage de résultats obtenus dans cet échantillon.

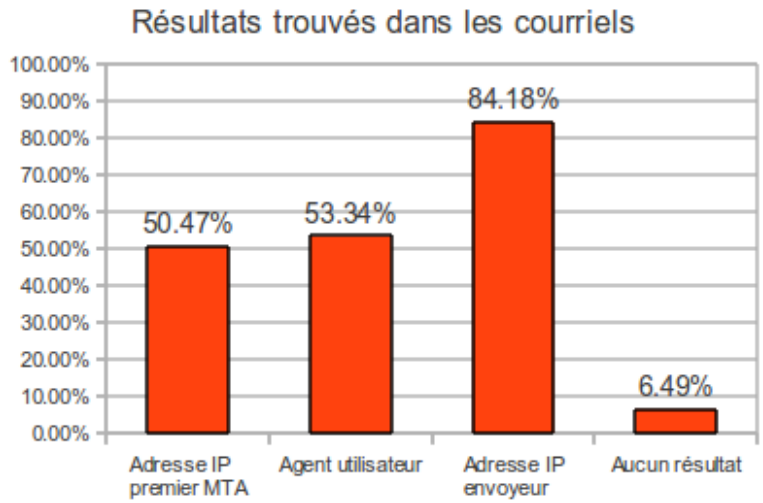


FIGURE 10.1 – Pourcentage de résultats sur un échantillon de données

EmailHeaderQualif a donc permis d'obtenir des informations sur plus de 73% des courriels de l'échantillon. Ce qui est une très bonne performance. Plus significatif, dans 84% des cas une adresse IP de machine émettrice est trouvée et dans 50% des cas une adresse IP de premier MTA a pu être trouvée. Ces adresses IP sont sujettes à interprétation car il n'est pas garanti qu'elles soient à 100% précises (certaines peuvent être forgées par exemple). Mais dans le cadre d'une investigation cela offre une capacité de recoupement non négligeable.

Dans un peu plus de la moitié des cas un agent utilisateur a pu être trouvé. Ce qui est une très bonne performance.

Pour les adresses IP de machine émettrice la répartition des résultats entre adresse ip privée, publique et aucun résultat est donnée dans la figure ci-après.

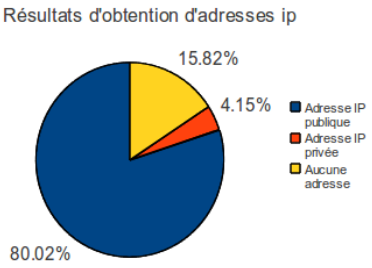


FIGURE 10.2 – Répartition des addresse IP source trouvées dans l'échantillon

Il y a 80% des cas ou une adresse IP privée est retournée. Ce qui est une très bonne performance. Le cas des adresses IP privées est de 4% ce qui est un apport assez minime mais qui peut quand même permettre des recoupement d'informations.

Finalement se trouve sur la page suivante, les résultats d'obtention d'informations sur les adresses IP publiques trouvées. Les valeurs des résultats correspondent à celles définies dans le cahier des charges.

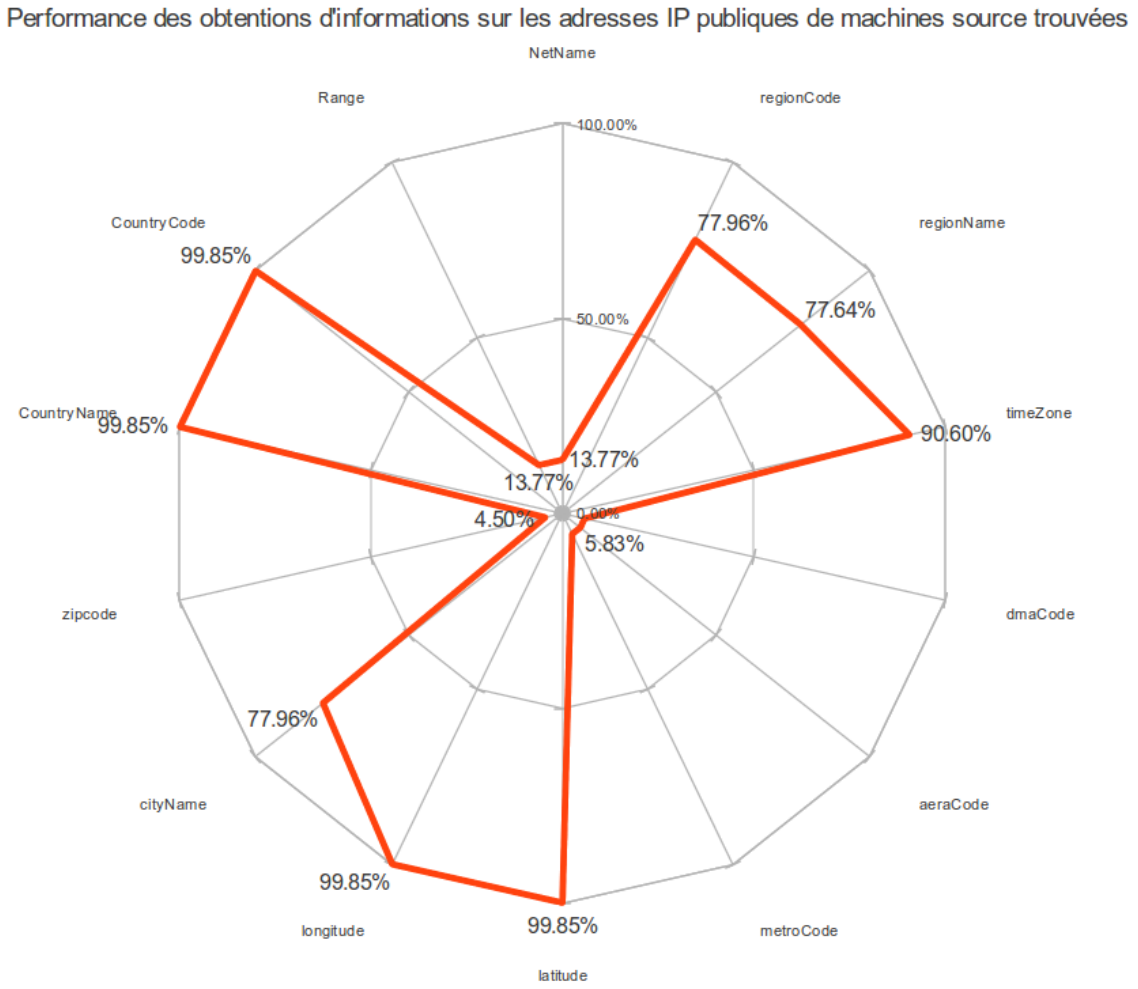


FIGURE 10.3 – Taux d'informations sur les adresses IP sources trouvées dans l'échantillon

Les performances de géo-localisation sont très bonnes. En effet dans presque 100% des cas l'adresse IP est localisée et ce au minimum dans le pays et dans 78% des cas dans une ville. Le restant des informations de géo-localisation dépends beaucoup de son emplacement. Par exemple «metroCode» qui ne peut être donné que pour des adresses aux USA.

Pour ce qui est des performances du module Whois correspondants aux valeurs «range» et «NetName», il y a 13% de résultats retournés par EmailHeaderQualif . Cette valeur peut être augmentée par la suite, car les données du module Whois utilisées sont celles obtenues après le problème de bannissement d'adresse IP publique sur les serveurs RIPE et ARIN (voir 8.2 pour les détails). Une fois ce problème résolu, le pourcentage de résultats peut augmenter significativement.

10.2 Intégration

L'intégration de EmailHeaderQualif dans ISCIA ou tout autre projet nécessite quelques points d'adaptation. Premièrement il faut contrôler la configuration de log4j pour la journalisation afin de rester en harmonie avec le projet dans lequel EmailHeaderQualif s'intègre. Deuxièmement avant d'intégrer le projet il faut résoudre le problème du bannissement d'adresse IP. Une solution est proposée dans 8.2 pour ce faire. Troisièmement il faut veiller à ce que les fichiers de configuration et de données des différents modules soient disponibles.

L'utilisation de EmailHeaderQualif se déroule en trois phase en utilisant la classe d'entrée EmailHeaderQualif :

- Appeler la méthode d'initialisation **start** pour initialiser les ressources de EmailHeaderQualif et les modules qui la compose.
- Utiliser la méthode headerQualif pour analyser les en-têtes de courriel
- Appeler la méthode de terminaison **stop** pour libérer les ressources utilisées par EmailHeaderQualif et les modules qui la compose.

10.3 Valeur ajoutée au projet ISCIA par EmailHeaderQualif

EmailHeaderQualif permet d'obtenir dans la foulée d'une identification de l'adresse IP source d'un courriel sa géo-localisation et l'intervalle auquel il appartient et son propriétaire. Ceci en garantissant une capacité de traitement supérieure à 100'000 en-têtes par jour.

La structure modulaire mise en place dans EmailHeaderQualif permet de rajouter facilement des modules supplémentaires pour obtenir des informations sur des en-têtes de courriels. Par ce biais, il est possible de réduire le coût de développement d'une nouvelle fonctionnalité concernant l'obtention d'informations à partir des en-têtes de courriels. Il est aussi possible d'activer et de désactiver les modules désirés à volonté via le fichier de configuration du gestionnaire de modules.

EmailHeaderQualif garantissant l'utilisation d'objets définis dans Java™ 1.6 dans ses retours et exceptions, le couplage entre ISCIA et EmailHeaderQualif ne tient que par les clés d'identification dans les différentes tables passées en paramètres. Ceci permet de presque complétement séparer ISCIA et EmailHeaderQualif , autant dans leur développement que dans leur utilisation.
De plus l'intégration du projet de Romain Cornu [14] n'est plus à faire dans ISCIA car elle a été faite dans ce projet.

Finalement le programme de test prévu initialement uniquement pour contrôler le fonctionnement de EmailHeaderQualif , peut être utilisé pour obtenir des statistiques sur des échantillon de courriels.

10.4 Déroulement du projet

Le déroulement du projet a été marqué par la modification fréquente des spécification et de la conception de ses différentes parties. C'est le résultat d'un manque de validation des spécification des différentes parties du projet. En effet la validation des spécifications de chacune des parties aurait permis de ne pas revenir dessus durant les entrevues avec le professeur responsable et de ne pas avoir à les modifier durant la réalisation. Mise à part ce point le projet s'est déroulé sans problèmes majeurs.

10.5 Difficultés du projet

Les difficultés techniques sont venues en premier de Maven. En effet la création et la compréhension d'un projet Maven a posé quelques soucis au départ. Cependant, la plus grosse difficulté a été de trouver les sources d'informations nécessaires lors de la pré-étude sur les botnets et sur les intervalles d'adresses IP. Un autre problème est survenu lors de la gestion des fichiers utilisés par le projet. En effet l'utilisation de Maven permet de définir un répertoire de ressources pour contenir les fichiers utilisés par le projet. Cependant, une fois la génération de l'archive JAR contenant le code compilé les fichiers mis dans ce répertoire ne sont plus accessibles via la classe «`File`» traditionnelle. Ils n'est pas possible non-plus d'écrire dans ces fichiers. Il a donc fallu mettre les fichiers de données hors de ce répertoire.

10.6 Apport personnel

Le déroulement de ce projet m'a permis de soulever un point que je n'ai que très peu eu durant ma formation : la validation par une personne externe. Il est clairement apparu durant ce projet que ce n'est de loin pas un réflexe chez moi. Pourtant, la valeur ajoutée par une validation d'une partie d'un projet est non-négligeable. Premièrement cela peut souvent éviter de se retrouver avec un problème qui aurait pu être évité si une personne externe avait porté un regard différent sur des spécifications, une analyse, ou encore une conception. Mais surtout, cela permet de gagner un temps précieux et de réduire le coût de développement.

D'une façon plus générale cela m'a permis de mettre en pratique les théories de gestion de projet et de constater leur importance. N'ayant travaillé que sur des projets cette notion de gestion n'avait jamais pris une telle importance pour moi.

Ce projet m'a aussi permis d'apprendre m'arrêter dans mon développement et me demander si ma façon de faire est la bonne lorsqu'un problème survient. C'est à dire de prendre un papier et un crayon pour essayer de résoudre un problème donné et moins fonctionner au cas par cas en continuant mon développement.

D'un point de vue technique ce projet m'a permis d'apprendre à utiliser Maven qui est un outil très puissant pour le développement et qui me semble maintenant presque indispensable à tout projet. Il m'a aussi donné la possibilité d'approfondir mes connaissances en Java et d'apprendre à utiliser correctement log4j pour la journalisation. J'ai aussi eu l'occasion d'améliorer mes connaissances sur les courriels. Cela va de la façon dont ils sont

gérés pour être transmis aux données contenues dans ceux-ci.

Yverdon-les-bains, le 16 décembre 2010 Nicolas Muller

Table des figures

1.1 Répartition des différents botnets (source [10]) octobre 2010 6
1.2 Vue globale du projet 12
1.3 Diagramme de Gant de la Planification 16

2.1 Schéma UML simplifié de l'analyseur modulaire 30
2.2 Diagramme d'activité de l'analyse d'une requête par le gestionnaire de modules 32

3.1 Schéma UML de IP Verification 43

4.1 Diagramme de classe simplifié du module de géo-localisation 48
4.2 Diagramme d'activité du *thread* de mise à jour des données de géo-localisation 49

5.1 Diagramme de classe simplifié du module user agent 54
5.2 Diagramme d'activité du *thread* de mise à jour des données de UserAgent . 55

6.1 Diagramme d'activité du *thread* de mise à jour du module Whois 62
6.2 Diagramme de classe simplifié du module Whois 64
6.3 Comparaison d'intervalles 70
6.4 Gestion des intervalles englobants 71
6.5 Diagramme d'activité de mise à jour des données whois 74
6.6 Diagramme de classe simplifié du code source «Whois» utilisé 76

7.1 Diagramme de classe du module IPBL 79

10.1 Pourcentage de résultats sur un échantillon de données 86
10.2 Répartition des addresse IP source trouvées dans l'échantillon 87
10.3 Taux d'informations sur les adresses IP sources trouvées dans l'échantillon . 88

D.1 Menu principal 111
D.2 Format d'affichage de la journalisation 114
D.3 Modification du niveau d'affichage de log4j 115

Glossaire

Notation	Description	Page List
CRLF	Cariage Return Line Feed : Indique un retour de chariot et un saut de ligne consécutif pour marquer la fin d'un en-tête et le début du message correspondant	3
ISCIA	Internet Surveillance for Criminal Intelligence Analysis	1, 8, 11
Nigerian Connection[7]	Arnaque connue qui utilise la crédulité des gens pour leur faire verser des sommes pour aider une personne (voir [7] pour plus d'informations	1
RMI	Remote Method Invocation : interface de programmation (API) pour le langage Java qui permet d'appeler des méthodes distantes.[9]	20
SpamHaus	Organisation de listage d'adresse IP	7
données locales	Données utilisées par les modules	29
géo-localisation	Module de géo-localisation d'une IP	16, 29
nslookup	*nslookup* est une commande permettant d'obtenir le nom DNS correspondant à une adresse IP	11
phishing	L'hameçonnage (ou phishing, et parfois filoutage), est une technique utilisée par des fraudeurs pour obtenir des renseignements personnels dans le but de perpétrer une usurpation d'identité. [2]	1
received	Éléments insérés dans un en-tête de courriel par chacun des MTA	10, 24
spam	Le spam, pourriel ou polluriel est une communication électronique non sollicitée, en premier lieu via le courrier électronique. Il s'agit en général d'envois en grande quantité effectués à des fins publicitaires.[8]	1

Notation	Description	Page List
user agent	Logiciel ou service web utilisé pour envoyer un courriel	1, 10, 11, 16, 22
API	Application Programm Interface : Interface de communication avec un programme par un programme tiers	8, 19
botnet	Un botnet est un ensemble de bots informatiques qui sont reliés entre eux. Historiquement, ce terme s'est d'abord confondu avec des robots IRC (bien que le terme ne se limitait pas à cet usage spécifique), qui était un type de botnet particulier servant sur les canaux IRC.	5
DNS	*Domain Name System* :	7
headerQualif	Nom de la méthode de qualification fournie pour le projet	10, 11, 14–17, 24, 25
IP	Une adresse IP (avec IP pour Internet Protocol) est un numéro qui est attribué à chaque appareil électronique lorsqu'il participe à un réseau informatique utilisant lorsqu'elle participe à un réseau informatique utilisant l'*Internet Protocol* comme protocole de communication entre ses nœuds. [1]	1, 3, 40
IPBL	Module de contrôle de validité d'adresse IP par liste noire (*Black List*	16

Notation	Description	Page List
Java™	Java est le nom de marque d'une technique développée par Sun Microsystems : la « technologie Java ». Elle correspond à plusieurs produits et spécifications de logiciels qui, ensemble, constituent un système pour développer et déployer des applications. Java est utilisée dans une grande variété de plates-formes depuis les systèmes embarqués et les téléphones mobiles jusque aux serveurs, les applications d'entreprise, les super ordinateurs et dans une moindre mesure pour les interfaces graphiques comme les applets Java du Web.	15, 40
JRE	Environnement d'exécution en machine virtuelle pour Java™	43
log4j	Système de journalisation pour Java™	15
MTA	Mail Transfer Agent : Agent de transfert de courriel. Se charge du transport des courriels entre l'envoyeur et le destinataire.	3, 4, 14, 39

Bibliographie

[1] Wikipédia *Internet Protocol*, *http* : *//fr.wikipedia.org/wiki/Adresse_ip*, 5 décembre 2010

[2] Wikipédia *Phishing/Hameçonnage*, *http* : *//fr.wikipedia.org/wiki/Phishing*, 12 décembre 2010

[3] Wikipédia *Java*, *http* : *//fr.wikipedia.org/wiki/Java_(technique)*, 10 octobre 2010

[4] Ceki Gülcü *Short introduction to log4j*, *http* : *//logging.apache.org/log4j/1.2/manual.html*, Mars 2002

[5] Sun Mycrosystems *Comment installer l'environnement d'exécution Java*, *http* : *//www.java.com/fr/download/help/download_options.xml*, dernière consultation novembre 2010

[6] Wikipédia *Whois*, *http* : *//fr.wikipedia.org/wiki/Whois*, 2 décembre 2010

[7] Christophe D'Antonio *Nigerian Connection*, *http* : *//www.ultrascan−agi.com/public_html/html/news/Nigerian_Connection.html*, 4 mars 2004

[8] Wikipédia *Spam*, *http* : *//fr.wikipedia.org/wiki/Spam*, 2 décembre 2010

[9] Wikipédia *Remote Method Invocation (Java)*, *http* : *//fr.wikipedia.org/wiki/Remote_method_invocation_(Java)*, 15 septembre 2010

[10] M86 Security Labs *Spam statistics*, *http* : *//www.m86security.com/labs/spam_statistics.asp*, octobre 2010

[11] The Spamhouse Project, *http* : *//www.spamhaus.org/index.lasso*, dernière consultation 20 novembre 2010

[12] Internet Engineering Task Force *RFC 1739 - A Primer On Internet and TCP/IP Tools*, *http* : *//www.faqs.org/rfcs/rfc1739.html*, Decembre 1994

[13] domaininformation.de *Whois server list*, *http* : *//www.domaininformation.de/whoisserver_list.html*

[14] Romain Cornu *Travail de Bachelor, Analyse de la source et du cheminement d'un e-mail*, 7 Aout 2010, HEIG-VD Yverdon-les-bains

[15] Wikipédia *RIPE*, *http* : *//en.wikipedia.org/wiki/RIPE*, 3 Novembre 2010

[16] ABDELALI Guerid, BREGUET Pierre, RÖTHLISBERGER Henri, «Algorithmes et structures de données avec Ada, C++ et Java», *Collection informatique*, 2004

[17] Oracle *Interface Comparable<T>*, *http : //download.oracle.com/javase/6/docs/api/java/lang/Comparable.html*, 2010

[18] Oracle *Class TreeMap<K,V>*, *http : //download.oracle.com/javase/6/docs/api/java/util/TreeMap.html*, 2010

[19] Free Software Foundation, Inc. *Gnu General Public Licence*, *http : //www.gnu.org/licenses/gpl.html*, 29 Juin 2007

[20] Oracle *Enum Thread.State*, *http : //download.oracle.com/javase/6/docs/api/java/lang/Thread.State.html*, 2010

[21] Frédéric Boussinot EMP-CMA/INRIA *Les threads*, *http : //www − sop.inria.fr/indes/rp/FairThreads/FTJava/french − presentation/index.html*, Octobre 2010

[22] Oracle *Interface XMLReader*, *http : //download.oracle.com/javase/6/docs/api/org/xml/sax/XMLReader.html*, 2010

[23] Oracle *Interface Serializable*, *http : //download.oracle.com/javase/6/docs/api/java/io/Serializable.html*, 2010

[24] Internet Engineering Task Force *RFC 5322 Internet Message Format*, *http : //tools.ietf.org/html/rfc5322*, avril 2001

[25] American Registry for Internet Numbers (ARIN) *Arin at glance*, *https : //www.arin.net/about_us/overview.html*

[26] DNSBL.info *Blacklists*, *http : //www.dnsbl.info/dnsbl − list.php*, 2010

[27] Wikipédia *Regional Internet registry*, *http : //en.wikipedia.org/wiki/Regional_Internet_registry*, Decembre 2010

[28] doug@neverfear.org *A Java WHOIS protocol implementation*, *http : //neverfear.org/blog/view/91/A_Java_WHOIS_protocol_implementation*, Janvier 2010

[29] W3C *Extensible Markup Language (XML) 1.0 (Fifth Edition)*, *http : //www.w3.org/TR/2008/REC − xml − 20081126/*, Novembre 2008

[30] AfriNIC *AfriNIC Database Reference Manual* , *http : //www.afrinic.net/docs/db/afsup − dbrefm200501.htm#17*, Janvier 2005

[31] RIPE NCC *Acceptable Use Policy*, *http : //www.ripe.net/db/support/db − aup.pdf*, dernière consultation le 12 décembre 2010

[32] APNIC *APNIC Whois Database Acceptable Use Agreement*, *http : //www.apnic.net/apnic − info/whois_search/about − whois/protecting − whois/copyright*, dernière consultation le 12 décembre 2010

[33] ARIN *WHOIS Terms of Use*, *https : //www.arin.net/whois_tou.html*, dernière consultation le 12 décembre 2010

[34] Rbot *Rbot trac page*, *http : //ruby − rbot.org/*, dernière consultation octobre 2010

[35] Wikipédia *Registre Internet régional*, *http : //fr.wikipedia.org/wiki/Registre_Internet_regional*, 12 novembre 2010

[36] Oracle *Class ClassLoader,http* : *//download.oracle.com/javase/6/docs/api/java/lang/ClassLoader.html*, 2010

[37] Internet Engineering Task Force *rfc 3912 - WHOIS Protocol Specification,http* : *//www.ietf.org/rfc/rfc*3912.*txt*, 2004

[38] Wikipédia *JUnit,http* : *//en.wikipedia.org/wiki/JUnit*, 2004

Annexe A

Résultats des tests de validation

Dans ce chapitre sont donnés les tableaux de résultats de la validation. Celle-ci a été faite par l'étudiant, le mandant et le professeur responsable.

A.1 Résultats de l'étudiant

N° cas test	Résultat	Remarque
1.3.3.1	OK/KO	PublicIPregionCode et PublicIPregionName disponnible en plus
1.3.3.2	OK	
1.3.3.3	OK/KO	
1.3.3.4	KO	Aucune qualifications car le module IPBL n'est pas implémenté
1.3.3.5	OK	
1.3.3.6	OK	
1.3.3.7	OK	
1.3.3.8	OK	
1.3.3.9	OK/KO	Il est apparu durant ces tests que les serveurs de ARIN et de RIPE ont bloqué les requêtes venant de l'adresse IP publique. Les tests ayant été fait exprès à domicile cela n'a aucun impact pour la HEIG-VD.
1.3.3.10	OK	

A.2 Résultats du professeur responsables

N° cas test	Résultat	Remarque
1.3.3.1	OK	
1.3.3.2	OK (KO)	Exception FileNotFound si on se trompe de nom de fichier
1.3.3.3	OK	

N° cas test	Résultat	Remarque
1.3.3.4	KO	IPBL pas implémenté => ne pas le faire tester
1.3.3.5	OK	
1.3.3.6	OK/KO	Indiquer les unités dans le programme de test
1.3.3.7	OK	
1.3.3.8	OK	
1.3.3.9	OK/KO	Pas testé à cause du problème des serveurs bloquant l'adresse IP. Comment faire pour ne pas être bloqué définitivement ?
1.3.3.10	KO	Pas de dates dans les logs, pas d'exception en entier dans le fichier de log

A.3 Résultats du mandants

N° cas test	Résultat	Remarque
1.3.3.1	OK	PublicIPregionCode et PublicIPregionName en plus
1.3.3.2	KO	Les trois mails avec AOL n'ont pas de user agent détectés
1.3.3.3	OK/KO	Il manque PublicIPQualification et FirstMTAIPQualification mais vu que IPBL non implémenté OK
1.3.3.4	OK	
1.3.3.5	OK	
1.3.3.6	OK	
1.3.3.7	OK	
1.3.3.8	OK	
1.3.3.9	OK	
1.3.3.10	OK	

Annexe B

Suivi du projet

N° Tâche	Tâche	Jours prévus	Temps prévu [h]	Temps prévu au 15.12.10 [h]	Temps effectué au 15.12.10 [h]	Différence
1.1	**Analyseur Modulaire**	**8**	**92.4**	**92.4**	**162.5**	**175.87%**
1.1.1	Spécifications	1	8.4	8.4	23.5	279.76%
1.1.2	Conception	2	16.8	16.8	13	77.38%
1.1.3	**Codage structure**	**3**	**25.2**	**25.2**	**44.5**	**176.59%**
1.1.3.1	Contrôleur de module	1	8.4	8.4	18	214.29%
1.1.3.2	Programme de test	2	16.8	16.8	26.5	157.74%
1.1.3.3	**Module IP Vérification**	**3**	**25.2**	**25.2**	**16**	**63.49%**
1.1.3.3.1	Extracteur d'IP	1	8.4	8.4	8	95.24%
1.1.3.3.2	Vérificateur d'IP	1	8.4	8.4	5	59.52%
1.1.3.3.3	Analyseur d'IP	1	8.4	8.4	3	100.00%
1.1.3.4	Module de qualification IP	1	8.4	8.4	0	100.00%
1.1.3.5	Intégration du module User Agent	1	8.4	8.4	11	130.95%
1.1.3.6	Intégration du module Géo-Localisation	1	8.4	8.4	13	154.76%
1.1.4	Tests unitaires	1	8.4	8.4	17.5	208.33%
1.1.5	Corrections	1	8.4	8.4	24	285.71%
1.2	**whois**	**7**	**58.8**	**58.8**	**91.5**	155.61%
1.2.1	Spécifications	0.5	4.2	4.2	5.5	130.95%
1.2.2	Conception	1	8.4	8.4	17	202.38%
1.2.3	Codage	3	25.2	25.2	42.5	168.65%
1.2.4	Tests unitaires	1.5	12.6	12.6	9.5	75.40%
1.2.5	Corrections	1	8.4	8.4	17	202.38%
1.3	**IPBL**	**5**	**42**	**42**	**10**	23.81%
1.3.1	Spécifications	0.5	4.2	4.2	3	71.43%
1.3.2	Conception	1	8.4	8.4	0	0.00%
1.3.3	Codage	2	16.8	16.8	0	0.00%
1.3.4	Tests unitaires	0.5	4.2	4.2	0	0.00%
1.3.5	Corrections	1	8.4	8.4	7	83.33%
1.4	**Rapport**	**12**	**100.8**	**100.8**	**183**	181.55%
1.4.1	Rapport analyseur	2	16.8	16.8	15	89.29%
1.4.2	Rapport IPBL	1	8.4	8.4	0	0.00%
1.4.3	Rapport whois	1	8.4	8.4	9.5	113.10%
1.4.4	Rapport intermédiaire	2	16.8	16.8	72	428.57%
1.4.6	Contrôle du code	1	8.4	8.4	6	71.43%
1.4.7	Finalisation rapport	5	42	42	80.5	191.67%
1.5	**Validation**	**2**	**16.8**	**16.8**	**2**	11.90%
1.5.2	Corrections validation analyseur	2	16.8	16.8	2	11.90%
1.6	**Suivi**	**2**	**16.8**	**16.8**	**23.25**	138.39%
1.6.1	Entrevues	1.5	12.6	12.6	10	79.37%
1.6.2	Doc Suivi	0.5	4.2	4.2	13.25	315.48%
	Total	**36**	**327.6**	**327.6**	**472.25**	**144.15%**

Annexe C

Documentation de développement

Dans ce chapitre ce trouvent quelques informations utile pour le développement de ce projet

C.1 Utilisation des paramètres de l'API en entrée

Les données reçues dans le header sont sous la forme d'une table de hashage. Cette table de hashage étend la classe **Map** de Java™. Dans cette table, les données et les clés sont en **String**. Chaque entrée correspond à une entrée «clé :valeur» de l'en-tête du courriel.

Les *receiveds* sont fournis dans une liste de table de hashage. Ils étendent eux aussi la classe **Map**.

Les clés des receiveds pris en entrée par l'API sont décrits ici. Ces champs sont toujours présents mais pas toujours renseignées (ils peuvent être à null).

Timestamp Date d'insertion du received

FromIp Adresse IP contenue dans le champ «from»

FromDns Nom DNS contenu dans le champ «from»

ByIp Adresse IP contenue dans le champ «By»

ByDns Nom DNS contenu dans le champ «by»

Via valeur de la clé «Via» dans le received

With valeur de la clé «With» dans le received

Id Numéro d'ID trouvé dans le receivd

For Destinataire trouvé dans le received

C.2 Structure des paquetages

Pour correspondre à la structure de paquetage de *ISCIA* les paquetages prennent le préfixe «ch.iict.iscia». Puis le préfixe «.ehq» est rajouté pour indiquer que c'est le paquetage de

EmailHeaderQualif .

Dans le paquetage «ch.iict.iscia.ehq» se trouve la classe EmailHeaderQualif . Toutes les autres classes se trouvent au minimum dans le paquetage «ch.iict.iscia.ehq.modules». Dans ce paquetage se trouve le code du gestionnaire de modules et les différents *enums* utilisés dans EmailHeaderQualif .

Le code de chaque module se trouve dans un paquetage «ch.iict.iscia.ehq.modules.NOMMODULE» où NOMMODULE est le nom du module.

C.3 log4j

Le système de journalisation log4j est utilisé par ce projet. Il est possible de modifier la façon dont est configuré log4j.

La configuration de log4j se fait à l'aide du fichier "log4j.properties" si le projet n'est pas inclus dans un autre qui prendrai en charge la journalisation.
Pour tout renseignements sur log4j se référer à [4]

C.3.1 Fichier POM pour la configuration de Maven

Le fichier POM.xml qui se trouve à la racine du projet permets de configurer le projet pour Maven. Il est configuré de telle manière à prendre le programme de test en classe d'entrée afin de pouvoir tester ou utiliser directement EmailHeaderQualif sans devoir passer par un autre programme. Il est aussi configuré pour générer deux fichiers JAR. Le premier qui ne contient pas les dépendances de EmailHeaderQualif telles que log4j. Et le deuxième auquel est rajouté «with-dependencies» qui inclut ces dépendances.

Le fichier est aussi configuré pour gérer les ressources utilisées par le projet. C'est à dire que le répertoire définit dans le fichier comme montré ci-après sera automatiquement inclus dans le fichier JAR. Les fichiers dans ce répertoire doivent être accédés par la méthode **getRessource** de la classe «`Class`».

```
<resources>
        <resource>
                <directory>
                        ${basedir}/src/main/resources
                </directory>
        </resource>
</resources>
```

Annexe D

Manuel utilisateur du programme de test

Ce manuel permet de mettre en place l'environnement approprié pour tester EmailHeaderQualif et explique les fonctions et l'utilisation du programme de test. Il faut savoir que la structure du programme de test a été reprise du programme de test fait par Romain Cornu dans son travail [14].

D.1 Installation de l'environnement

Pour pouvoir lancer les tests il faut disposer du JREet du fichier jar contenant le programme de test et l'API complète. Dans la suite de cette section sont donnés les méthodes d'installation du JRE

D.1.1 Installation du JRE

Cette section s'inspire du cite officiel java[5]. Il sera décrit ici, de façon simplifiée, l'installation pour Windows et Linux du JRE. Pour plus d'informations se référer à [5]
. Vous devez disposer des droits administrateur pour pouvoir faire cette installation.

D.1.1.1 Installation sous Windows

1. Aller sur la page `http://www.java.com/fr/download/manual.jsp` et télécharger la version 6.x du JRE correspondant à votre système d'exploitation
2. Double cliquer sur le fichier téléchargé
3. Accepter les termes du contrat de licence
4. L'installeur peut proposer des programmes supplémentaires. Ils ne sont pas nécessaires au bon fonctionnement du programme de test. Il n'est donc pas nécessaire de les sélectionner pour installation
5. À la fin de l'installation cliquer sur «fermer» pour terminer l'installation

D.1.1.2 Installation sous Linux

Sous Ubuntu :

– Ouvrir un terminal
– Exécuter **sudo apt-get install sun-java6-jre** pour la version propriétaire du JRE ou **sudo apt-get install openjdk-6-jre** pour la version libre.

Pour plus de détails voir `http://doc.ubuntu-fr.org/java`

Pour toutes les autres distributions Linux :

1. Aller sur la page `http://www.java.com/fr/download/manual.jsp` et télécharger la version 6.x du JRE correspondant à votre système d'exploitation
2. Déplacer le fichier d'installation dans le répertoire cible d'installation
3. Ouvrir un terminal
4. Exécuter **cd <rep_téléchargement>** où <rep_téléchargement> est le répertoire où se trouve le fichier exécutable d'installation du JRE
5. Exécuter **su**
6. Exécuter **chmod a+x <nom_fichier>.bin** où <nom_fichier> est le nom du fichier exécutable
7. Exécuter **./<nom_fichier>.bin** ce qui va lancer l'installation
8. Exécuter **pwd** le répertoire qui vous est donné est le répertoire d'installation du JRE

D.2 Manuel d'utilisation

Le programme de test de EmailHeaderQualif a plusieurs fonctionnalités. Il permet de tester le lancement et l'arrêt de EmailHeaderQualif . Il permet de faire des tests de EmailHeaderQualif en se basant sur des fichiers contenant des en-têtes types et il permet de créer ces fichiers de données à partir de fichiers texte préformatés. Le programme de test est à utiliser en ligne de commande et ne possède aucune interface graphique.

Se trouvent dans la suite de ce manuel :
– Méthode de lancement du programme et premier contact
– Description détaillée des fonctionnalités du programme et de leur utilisation
– Moyen d'analyse des résultats
– Modification de l'affichage de la journalisation
– Erreurs connues pouvant survenir

D.2.1 Lancement du programme de test et utilisation

Premièrement vérifier que le fichier GeoLiteCity.dat est bien présent dans le répertoire où est exécuté le programme de test.
Dans le terminal lancer la commande **java -jar <chemin du fichier jar>**, ce qui aura pour effet de lancer le programme de test. Apparaissent alors les informations suivantes :

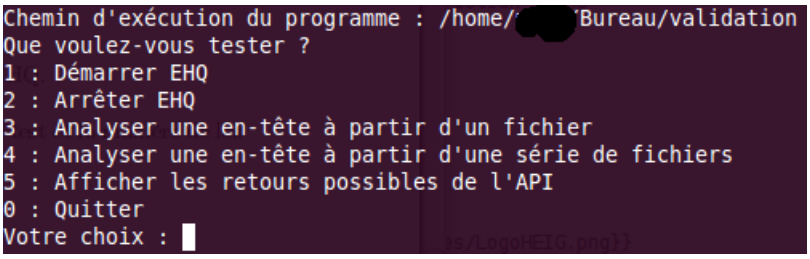

Chemin d'exécution du programme : /home/ Bureau/validation
Que voulez-vous tester ?
1 : Démarrer EHQ
2 : Arrêter EHQ
3 : Analyser une en-tête à partir d'un fichier
4 : Analyser une en-tête à partir d'une série de fichiers
5 : Afficher les retours possibles de l'API
0 : Quitter
Votre choix :

FIGURE D.1 – Menu principal

La première ligne donne le répertoire d'exécution du programme. Cette information s'avère utile quand il faut entrer des chemins de fichier pour les différents tests.

Il est à savoir que le programme tourne en boucle. Peu importe ce qui se passe dans une des fonctionnalités du programme. Il peut même survenir une exception le programme va l'afficher et retourner au menu principal. Quand une fonctionnalité est terminée le programme revient automatiquement au menu initial.

Comme on peut le voir EmailHeaderQualif est abrégé ici en EHQ.

La section suivante décrit les fonctionnalités du programme de test et la manière de les utiliser.

D.2.2 Fonctionnalités du programme de test

Les différentes fonctionnalités présentées dans la suite de cette section sont les suivantes (les numéros correspondent au menus du programme) :

1. Démarrage de EmailHeaderQualif
2. Arrêt de EmailHeaderQualif
3. Analyse d'un en-tête contenu dans un fichier par EmailHeaderQualif
4. Analyse des en-tête contenus dans tous les fichiers d'un dossier
5. Affichage des retours possibles

D.2.2.1 Démarrage de EmailHeaderQualif

L'option «1» du programme permet d'appeler la méthode *start* de EmailHeaderQualif et d'initialiser le gestionnaire de modules et les différents modules. Une fois cette option choisie l'initialisation est lancée et les différentes séquences d'initialisation sont affichées. Puis le programme retourne au menu principal.

D.2.2.2 Arrêt de EmailHeaderQualif

L'option «2» permet d'appeler la méthode *stop* de EmailHeaderQualif ce qui stoppe chacun des modules pour libérer les ressources et stoppe le gestionnaire de module.

D.2.2.3 Analyse d'un en-tête contenu dans un fichier par EmailHeaderQualif

Cette fonction a pour but de lancer un test à partir d'un fichier texte contenant un courriel entier (en-tête et corps). Il faut sélectionner l'option «3» pour pouvoir l'utiliser.

Un fois cette option choisie un chemin du fichier de test est demandé. Ce chemin peut être relatif ou absolu.Par exemple, si le programme s'exécute dans le répertoire */home* et que le chemin du fichier de test est */home/fichierTest/test.txt*. Il est possible de rentrer le chemin absolu */home/fichierTest/test.txt* ou le chemin relatif *fichierTest/test.txt*.

Une fois le fichier de configuration sélectionné, l'en-tête contenu dans ce fichier est affiché sous forme structurée. Puis le programme appelle la méthode headerQualif de EmailHeaderQualif pour analyser cet en-tête. Une fois l'analyse terminée le résultat est affiché, toujours sous forme «clé :valeur» correspondant à la table de hachage retournée. De plus est affiché le temps d'exécution de l'analyse.

D.2.2.4 Analyse des en-tête contenus dans tous les fichiers d'un dossier

L'analyse de tout les fichiers contenu dans un dossier est semblable à D.2.2.3 sauf qu'elle s'exécute sur tout les fichiers contenus dans un dossier. Le numéro de menu correspondant à cette fonction est le «4».

Une fois cette option choisie un chemin du dossier contenant les fichiers de test est demandé. Ce chemin peut être relatif ou absolu.Par exemple, si le programme s'exécute dans le répertoire */home* et que le chemin du dossier est */home/fichierTest*. Il est possible de rentrer le chemin absolu */home/fichierTest* ou le chemin relatif *fichierTest*.

Une fois le répertoire de test sélectionné, le programme de test va parcourir tous les fichiers et pour chacun d'eux lancer une analyse avec la méthode headerQualif de EmailHeaderQualif . Avant d'être analysé, le chemin du fichier contenant l'en-tête est affiché. Puis l'analyse est faite et finalement le résultat est affiché ainsi que le temps d'exécution pour le fichier.

À la fin est affiché, en secondes, le temps d'exécution total pour l'ensemble des fichiers, la moyenne d'exécution par fichier, le temps maximum d'exécution par fichier et le temps minimum d'exécution. Ces temps représentent le temps de réponse de la méthode headerQualif .

D.2.2.5 Affichage des retours possibles de EmailHeaderQualif

L'option «5» permet d'afficher la liste des retours possibles de la méthode headerQualif de EmailHeaderQualif .

D.2.2.6 Terminaison du programme

En sélectionnant l'option «0» le programme sort de sa boucle infinie. Attention toutefois cela n'appelle pas la méthode *stop* de EmailHeaderQualif . Les ressources ne sont donc pas

libérées correctement. Cela étant le programme de test peut ne pas se terminer car un des threads reste actif.

D.2.3 Analyse des résultats et des informations affichées

Lors de l'utilisation des différentes fonctionnalités du programme, plusieurs informations sont affichées. Comme cité en D.2.2.3 et D.2.2.4 des temps d'exécution sont affichés. Mais aussi les en-têtes analysés et les résultats obtenus. De plus, des informations sont affichées durant l'appel aux fonctions de EmailHeaderQualif sur son exécution. Ci-après sont donnés le format des en-têtes affichés et des résultats, ainsi que l'explication des différentes informations affichées.

D.2.3.1 Données affichées

Deux types de données sont affichés : les en-têtes extraits des fichiers de test et les résultats des appels à la méthode headerQualif . Ils sont expliqués ci-dessous.

D.2.3.1.1 Données dans les en-têtes Un en-tête est affiché dans le format suivant :

```
{header :
  {cle1=valeur, cle2=valeur, clex=valeur\},
 receiveds :
  [{ByDns=, ByIp=, For=, FromDns=, FromIp=, Id=, Timestamp=, Via
      =, With=},
   {ByDns=, ByIp=, For=, FromDns=, FromIp=, Id=, Timestamp=, Via
      =, With=}
  ]
}
```

On peut voir la liste des valeurs contenues dans l'en-tête puis la liste des receiveds du dernier inséré au premier inséré.

D.2.3.1.2 Résultat des appels à l'API Les résultats sont affichés selon le schéma suivant :

```
{Cle1=valeur, cle2=valeur, cleX=valeur}
```

Chacune des valeurs possibles de retour est donnée dans le cahier des charges.

D.2.3.2 Informations d'exécution de l'API

Durant l'exécution du programme de test, des informations sont affichées durant les appels à la méthode headerQualif . Plusieurs niveaux d'informations sont donnés en se basant sur log4j et le format d'affichage est un peu spécial. Ils sont tous les deux expliqués dans la suite de cette section.

D.2.3.2.1 Niveau d'information Le programme de test se base sur les logs de EmailHeaderQualif pour faire l'affichage des informations d'exécution. log4j étant utilisé pour faire cette journalisation, il est important de comprendre son fonctionnement de base pour pouvoir en faire une interprétation. C'est ce qui est expliqué ici. Plus d'informations sur log4j peuvent être trouvées sur [4].

log4j utilise des "niveaux" de journalisation. La différence entre ces niveaux vient de leur importance. Ci après se trouve une brève description de chacun d'eux [1] (de la priorité la plus haute à la plus basse) :

OFF Niveau le plus haut de journalisation qui a pour but d'arrêter celle-ci

FATAL Désigne une erreur fatale qui oblige l'application à s'arrêter

ERROR Ce niveau indique les événements d'erreur qui peuvent survenir mais qui n'empêche pas forcément l'application à s'arrêter

WARN Journalisation de situation qui peuvent être critique pour le bon déroulement de l'application

INFO Niveau d'information du programme qui permet de suivre le déroulement

DEBUG Niveau d'information plus fin que INFO utilisé pour le débogage des applications

TRACE Désigne un niveau d'information légèrement supérieur à DEBUG. Par exemple, il affiche toutes les exceptions qui surviennent même si elles ne sont pas traitées.

ALL Journalise toutes les informations de débogage

D.2.3.2.2 Format d'affichage Le format d'affichage est le suivant :

06:12:47 [INFO] [IPAnalyserModule] (start:111): Starting EHQ modules
06:12:47 [INFO] [IPAnalyserModule] (start:112): Loads modules list
06:12:47 [INFO] [IPAnalyserModule] (start:114): Builds dependencies
06:12:47 [INFO] [IPAnalyserModule] (start:116): Starts modules
1
2
3
4
5

FIGURE D.2 – Format d'affichage de la journalisation

Le premier élément (1) est l'heure de l'entrée de journalisation. Ensuite (2) se trouve, entre crochets, le niveau de journalisation. À la suite (3) se trouve, entre crochets, le nom de la classe (paquetage compris) d'où vient cette journalisation puis (4) la méthode dans la classe ainsi que la ligne de code entre parenthèses. Finalement (5) se trouve la ligne de texte de l'entrée de journal.

D.2.4 Modification de l'affichage des informations

Il est possible de changer le niveau d'affichage des informations dans le ficher «log4j.properties». Il suffit de changer la valeur encadrée en rouge dans la figure D.3. Les valeurs qui peuvent être mises ici correspondent au niveau existant dans log4j (voir paragraphe précédent pour le détail des niveaux).

Plus les priorités baissent, plus la quantité d'informations affichée est grande.

1. source : `http://logging.apache.org/log4j/1.2/apidocs/org/apache/log4j/Level.html`

```
#Level and appenders logger définition
log4j.rootLogger=INFO, stdout

#stdout appender configuration
log4j.appender.stdout=org.apache.log4j.ConsoleAppender

#layout définition for stdout
log4j.appender.stdout=org.apache.log4j.ConsoleAppender
log4j.appender.stdout.layout=org.apache.log4j.PatternLayout
log4j.appender.stdout.layout.ConversionPattern=%d{hh:mm:ss}
```

FIGURE D.3 – Modification du niveau d'affichage de log4j

D.2.5 Erreurs connues pouvant survenir

Certaines erreurs connues peuvent survenir durant l'utilisation du programme de test. Ici sont détaillées les erreurs connues ou fréquentes et ce qui peut être fait pour les contrer.

Erreur de mise à jour de GeoIP Il est possible qu'une erreur «GeoIP dataSource has been not updated error while update» puisse survenir durant le téléchargement de la base de donnée de géo-localisation d'adresse IP. En effet lors de téléchargements répétés le site ou se trouve le fichier de donnée bloque le téléchargement pour quelques heures. Cela n'empêche aucunement le fonctionnement de EmailHeaderQualif . Pour résoudre ce problème il faut soit trouver un moyen de changer de connexion internet (changement d'adresse IP) soit attendre que le téléchargement ne soit plus bloqué.

Annexe E

DVD